

VisualAge Generator



Web Transaction Development Guide

Version 4.5

VisualAge Generator



Web Transaction Development Guide

Version 4.5

Note

Before using this document, read the general information under “Notices” on page v.

First Edition (September 2000)

This edition applies to the following licensed programs:

- IBM VisualAge Generator Developer for OS/2 and Windows NT Version 4.5
- IBM VisualAge Generator Server for OS/2, AIX, Windows NT, HP-UX, and Solaris Version 4.5
- IBM VisualAge Generator Server for AS/400 Version 3.1
- IBM VisualAge Generator Server for AS/400 Version 3.6
- IBM VisualAge Generator Server for MVS, VSE, and VM Version 1.2

Order publications by phone or fax. IBM Software Manufacturing Solutions takes publication orders between 8:30 a.m. and 7:00 p.m. eastern standard time (EST). The phone number is (800) 879-2755. The fax number is (800) 445-9269. Faxes should be sent Attn: Publications, 3rd floor.

You can also order publications through your IBM representative or the IBM branch office serving your locality. Publications are not stocked at the address below.

IBM welcomes your comments. You can send your comments in any one of the following methods:

Electronically, using the online reader comment form at the address listed below. Be sure to include your entire network address if you wish a reply.

- <http://www.ibm.com/software/ad/visgen>

By mail to the following address:

IBM Corporation, Attn: Information Development, Department G7IA Building 062, P.O. Box 12195, Research Triangle Park, NC 27709-2195.

When you send information to IBM, you grant IBM a nonexclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

© Copyright International Business Machines Corporation 2000. All rights reserved.

US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

Notices.	v	
Trademarks.	vii	
Terminology used in this document	ix	
Terminology differences between Java and Smalltalk	x	
About This Document	xi	
Documentation provided with VisualAge Generator	xi	
Chapter 1. Introduction	1	
How Segmented Programs Work	1	
UI record	2	
Runtime architecture of VisualAge Generator web transactions	3	
Tier 1	4	
Tier 2	5	
Tier 3	9	
CONVERSE and XFER	10	
CONVERSE	10	
XFER with a named program	12	
XFER with a blank.	14	
How Input Data is Edited	16	
Chapter 2. UI Types	19	
Basic HTML Structure.	21	
Default <FORM> structure	23	
Non-default <FORM> structure	24	
Web-page access to other generated programs	24	
Control Items	27	
HTML from UI type Input or Input/Output	28	
The effect of a Boolean edit check.	28	
The effect of data-item length	29	
The effect of a UI item array	30	
The effect of a selected index item (occurs=1).	31	
The effect of a selected index item (occurs > 1).	32	
The effect of a match-valid table	34	
HTML from UI type Output	34	
An item of UI type Output	34	
The effect of a UI item array	34	
The effect of a selected index item (occurs = 1).	35	
The effect of a selected index item (occurs > 1).	36	
The effect of substructuring an array.	37	
UI type None	40	
UI types Submit and Submit Bypass	40	
How values and labels are translated	40	
How to render an image in place of an HTML SUBMIT button	41	
The effect of parent UI items	42	
Help text	42	
Web page attribute setting	42	
Data item edits	43	
Chapter 3. Resource bundles	45	
Resource bundles at runtime	45	
Resource bundles at generation time	46	
User-defined messages	46	
Chapter 4. Testing Web Transactions	49	
ITF	49	
Callable ITF	50	
Assigning a TCP/IP port to Callable ITF	51	
Configuring the gateway servlet	52	
Identifying and deploying tier 2 objects	53	
Author-Time Values	53	
WebSphere Test Environment	54	
Configuring the test environment	55	
Running a web transaction in WTE	61	
Chapter 5. Java Server Pages and the UI Record Bean API	63	
JSP syntax.	63	
Scriptlets	63	
Expressions	64	
Bean tag	65	
Directives	66	
The UI record bean API	67	
UI Record Bean Interface.	67	
VGDataElement Interface	68	
Chapter 6. Reviewing Sample Web Transactions	71	
Set up	71	
Extracting the sample files	71	
Setting up the database	71	

Loading the code	72
Program structure	73
CONVERSE	73
XFER with a named program	76
XFER with a blank.	83
Implementing global state management.	90
UI record-based state management implementation	91
Working-storage-record-based state management	103
Self-managed state implementation (XFER with a blank)	107
Chapter 7. Running Web Transactions in WebSphere Application Server.	111
Configuring a web app	111
Define an application server, servlet engine, and web application to WAS	111
Configure the web application	112
Complete the general set up	113
Running a web transaction in WAS	113
Start the Session ID Manager	114
Start the web application server	114
Invoke the gateway servlet.	114
Start a catcher program on tier 3.	115
Maintaining the work database on tier 3	115

Appendix A. Configuring the Gateway

Servlet	117
Configuring a Gateway Servlet	117
hptLogonPage Parameter	118
hptEntryPage Parameter	119
hptLinkageProperties Parameter	119
hptServletVersion Parameter	124

Appendix B. Transforming TUIs into Web

Transactions	127
Considerations.	127
Phase 1 — analysis	127
Phase 2 — basic transformation	128
Prepare the code to be transformed	128
Convert the map to a UI record and replicate code	128
Implement UI record processing using CONVERSE.	129
Transform existing logic.	130
Test your program	130
Phase 3 — customization for the web	131
Phase 4 — modify default JSP	132

Index	133
------------------------	------------

Notices

References in this publication to IBM products, programs, or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Subject to IBM's valid intellectual property or other legally protectable rights, any functionally equivalent product, program, or service may be used instead of the IBM product, program, or service. The evaluation and verification of operation in conjunction with other products, except those expressly designated by IBM, are the responsibility of the user.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to the IBM Director of Licensing, IBM Corporation, 500 Columbus Avenue, Thornwood NY 10594, U.S.A.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact the SWS General Legal Counsel, IBM Corporation, Department TL3 Building 062, P. O. Box 12195, Research Triangle Park, NC 27709-2195. Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

IBM has made reasonable efforts to ensure the accuracy of the information contained in this publication. If a softcopy of this publication is provided to you with the product, you should consider the information contained in the softcopy version the most recent and most accurate. However, this publication is presented "as is" and IBM makes no warranties of any kind with respect to the contents hereof, the products listed herein, or the completeness or accuracy of this publication.

IBM may change this publication, the product described herein, or both.

Trademarks

The following terms are trademarks of the IBM Corporation in the United States or other countries or both:

AD/Cycle
AIX
AIX/6000
AS/400
CICS
CICS/ESA
CICS OS/2
CICS/MVS
CICS/VSE
CICS/6000
COBOL/370
COBOL/400
C Set + +
DB2
DB2/2
DB2/400
Distributed Relational Database Architecture
DRDA
IBM
IMS
IMS/ESA
Language Environment
MQSeries
MQ
MVS
MVS/ESA
OS/2
OS/390
OS/400
Operating System/2
OpenEdition
PS/2
Presentation Manager
RACF
RISC System/6000
S/390
SAA
SQL/DS
SQL/400

VisualAge
VisualGen
Virtual Machine/Enterprise Systems Architecture
VM/ESA
VSE/ESA
VTAM

The following terms are trademarks of other companies:

Attachmate	Attachmate Corporation
Borland	Borland International, Inc.
HP	Hewlett-Packard Company
HP-UX	Hewlett-Packard Company
Intel	Intel Corporation
Interspace	Planetworks L.L.C.
NetWare	Novell, Incorporated
Novell	Novell, Inc.
PC-DCE/32	Gradient Technologies, Inc.
PIC	Pacific Image Communications, Incorporated
Planetworks	Planetworks L.L.C.
PowerBuilder	Sybase, Inc.
Tandem	Tandem Computers Incorporated
VMS	Digital Equipment Corporation
X/Open	X/Open Company Limited

Microsoft, Windows, Windows NT, Windows 2000, the Windows logo, and the Windows 95 logo the are trademarks of Microsoft Corporation in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Solaris, Java, and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Other company, product, or service names may be the trademarks or service marks of others.

Terminology used in this document

Unless otherwise noted in this publication, the following references apply:

- MVS CICS applies to Customer Information Control System/Enterprise Systems Architecture (CICS/ESA) systems.
- CICS applies to CICS for VSE/ESA, CICS/ESA, CICS for OS/2, CICS for AIX, CICS for Windows NT, and CICS for Solaris.
- CICS for Windows NT refers to IBM TXSeries for Windows NT Version 4.2.
- CICS for AIX refers to IBM TXSeries for AIX Version 4.2.
- CICS for Solaris refers to IBM WebSphere Enterprise Edition Version 3.0.
- IMS/VS applies to Information Management System/Enterprise System Architecture (IMS/ESA) and IMS/ESA Transaction Manager systems.
- IMS applies to IMS/ESA and IMS/ESA Transaction Manager, and to message processing program (MPP), IMS Fast Path (IFP), and batch message processing (BMP) regions. IMS/VS is used to distinguish MPP and IFP regions from the IMS BMP target environment.
- LE applies to the IBM Language Environment for MVS and VM.
- COBOL applies to any of the following types of COBOL:
 - IBM VisualAge for COBOL for OS/2
 - ILE COBOL/400
 - IBM COBOL for VSE
 - IBM COBOL for MVS and VM
- “Region” and “CICS region” correspond to the following:
 - CICS for MVS/ESA region
 - IMS region
 - CICS for VSE/ESA partition
 - CICS for OS/2 system
 - CICS for AIX system
 - CICS for Windows NT system
 - CICS for Solaris system
- DB2/VSE refers to SQL/DS Version 3 Release 4 or later. Any references to SQL/DS refer to DB2/VSE and SQL/DS on VM. In addition, any references to SQL/400 refer to DB2/400.
- OS/2 CICS applies to CICS Operating System/2 (CICS for OS/2).
- Workstation applies to a personal computer, not an AIX workstation.
- The make process applies to the generic process not to specific make commands, such as make, nmake, pmake, polymake.
- Unless otherwise noted, references to VM apply to Virtual Machine/Enterprise Systems Architecture (VM/ESA) environments.
- References to VM batch apply to any batch facility running on VM.
- DB2/2 applies to DB2/2 Version 2.1 or later, and DB2 Universal Database (UDB) for OS/2 Version 5.

- DB2/6000 applies to DB2/6000 Version 2.1 or later, and DB2 Universal Database (UDB) for AIX Version 5.
- Windows applies to Windows 95, Windows 98, Windows NT, and Windows 2000.
- Unless a specific version of Windows NT is referenced, statements regarding Windows NT also apply to Windows 2000.

Terminology differences between Java and Smalltalk

VisualAge Generator Developer can be installed as a feature of VisualAge for Java or VisualAge Smalltalk. Where appropriate, the documentation uses terminology that is specific to Java or Smalltalk. But where the information is specific to VisualAge Generator and virtually the same for both environments, the Java/Smalltalk term is used.

Table 1. Terminology differences between Java and Smalltalk

Java term	Combined Java/Smalltalk term	Smalltalk term
Project	Project/Configuration map	Configuration map
Package	Package/Application	Application
Workspace	Workspace/Image	Image
Beans palette	Beans/Parts palette	Parts palette
Bean	Visual part or bean	Visual part
Repository	Repository/ENVY library	ENVY library manager
Options	Options/Preferences	Preferences

About This Document

This document introduces you to the VisualAge Generator technology for developing, testing, and deploying web-based applications that access resources on server platforms. Many of the techniques and concepts described are familiar to VisualAge Generator developers, who can use existing code as a starting point when writing programs that provide service on the Internet.

By separating the task of designing a user interface from the task of writing business logic, VisualAge Generator allows an organization to more effectively use the skills of web designers and enterprise developers. By hiding implementation details and providing special kinds of testing support, VisualAge Generator increases the productivity of both kinds of personnel.

This document does the following:

- Describes the architecture of a web-based application.
- Tells how existing VisualAge Generator language elements are used in web development.
- Relates how to use a UI record to create a prototype web page.
- Covers the test options that help you move a web-based application from prototype to production.
- Reviews training examples that are installed with VisualAge Generator.
- Identifies the procedures for deploying a web-based application to WebSphere Application Server version 3.5.

Additional material is available from web site <http://www.redbooks.ibm.com>:

- *Servlet and JSP Programming with IBM WebSphere Studio and VisualAge for Java* (SG24-5755-00) is a detailed introduction to servlets and Java Server Pages (JSPs).
- *Building Enterprise Web Transactions using VisualAge Generator JavaBeans and JSPs* (SG24-5636-00) guides you in customizing JSPs that are produced with VisualAge Generator and gives details on setting up a Windows-based environment that includes TXSeries (CICS), DB2, and PCOMM.

Documentation provided with VisualAge Generator

VisualAge Generator documents are provided in one or more of the following formats:

- Printed and separately ordered using the individual form number.

- Online book files (.pdf) on the product CD-ROM. Adobe Acrobat Reader is used to view the manuals online and to print desired pages.
- HTML files (.htm) on the product CD-ROM and from the VisualAge Generator web page (<http://www.ibm.com/software/ad/visgen>).

The following books are shipped with the VisualAge Generator Developer CD. Updates are available from the VisualAge Generator Web page.

- *VisualAge Generator Getting Started* (GH23-0258-01) ^{1,2}
- *VisualAge Generator Installation Guide* (GH23-0257-01) ^{1,2}
- *Introducing VisualAge Generator Templates* (GH23-0272-01) ^{2,3}

The following books are shipped in PDF and HTML formats on the VisualAge Generator CD. Updates are available from the VisualAge Generator Web page. Selected books are available in print as indicated.

- *VisualAge Generator Client/Server Communications Guide* (SH23-0261-01) ^{1, 2}
- *VisualAge Generator Design Guide* (SH23-0264-00) ¹
- *VisualAge Generator Generation Guide* (SH23-0263-01) ¹
- *VisualAge Generator Messages and Problem Determination Guide* (GH23-0260-01) ¹
- *VisualAge Generator Programmer's Reference* (SH23-0262-01) ¹
- *VisualAge Generator Migration Guide* (SH23-0267-00) ¹
- *VisualAge Generator Server Guide for Workstation Platforms* (SH23-0266-01) ^{1,4}
- *VisualAge Generator System Development Guide* (SG24-5467-00) ²
- *VisualAge Generator User's Guide* (SH23-0268-01) ^{1, 2}
- *VisualAge Generator Web Transaction Development Guide* (SH23-0281-00) ¹

The following documents are available in printed form for VisualAge Generator Server for AS/400 and VisualAge Generator Server for MVS, VSE, and VM:

- *VisualAge Generator Server Guide for AS/400* (SH23-0280-00) ²
- *VisualAge Generator Server Guide for MVS, VSE, and VM* (SH23-0256-00) ²

The following information is also available for VisualAge Generator:

- *VisualAge Generator External Source Format Reference* (SH23-0265-01)
- *Migrating Cross System Product Applications to VisualAge Generator* (SH23-0244-01)
- *VisualAge Generator Templates V4.5 Standard Functions—User's Guide* (SH23-0269-01) ^{2, 3}

1. These documents are available as HTML files and PDF files on the product CD.

2. These documents are available in hardcopy format.

3. These documents are available as PDF files on the product CD.

4. This document is included when you order the VisualAge Generator Server product CD.

Chapter 1. Introduction

The VisualAge Generator technology for creating web-based applications is built on a code base that has long supported the development of *segmented programs*—programs that maintain a conversation with a user across several invocations. This chapter first describes what happens when a segmented program runs, then provides details on VisualAge Generator support for a new type of program, the *web transaction*, which runs in many cases as a segmented program.

How Segmented Programs Work

A segmented program does the following when the user first invokes it:

- Performs initialization tasks.
- Saves *program state*—a set of user-specific values that reflects the current status of the user-program conversation.
- Releases database locks and other resources.
- Presents one of several screens.
- Ends.

The user's next invocation seems to be a continuation of processing; the user clicks a button, for example, to update business data. In fact, however, the program returns to memory, restores program state, processes the user's input, and continues the cycle by saving program state, releasing resources, sending output to the user, and ending.

On environments that support segmented programs, VisualAge Generator accesses a *work database* to store and retrieve program state. For details, see "Maintaining the work database on tier 3" on page 115.

In the absence of VisualAge Generator, the developer of a segmented program writes code to analyze program state at each invocation. With VisualAge Generator, however, the application logic is simpler because the developer writes code as if the user were having an ongoing conversation with a program that is always in memory.

The developer can use a segmented VisualAge Generator program as a starting point when developing a web transaction because the processing of the legacy and web programs are similar:

- A mainframe program, for example, sends a 3270 data stream to each of several terminals; a web transaction sends an HTML data stream to each of several browsers.
- The legacy program relies on middleware to interact with multiple users simultaneously; the web transaction relies on a web application server to interact with multiple users simultaneously.
- The legacy program saves and restores program state in a way that is transparent to the developer, the web transaction does the same.
- The legacy developer can select among the following programming models, as can the web-transaction developer:
 - In the *complete-state* model, VisualAge Generator saves all data when a program ends processing in the middle of a conversation. This model is implemented by the I/O option CONVERSE.
 - In the *controlled-state* model, the developer determines what state data to save. This model is implemented by the XFER statement.

The web-transaction developer can select the *stateless* model as well, in which case no state data is saved. This model is also implemented by the XFER statement.

Support for the controlled state and stateless models gives the web developer maximum flexibility. This support is particularly valuable for reducing the disk-storage requirements that otherwise would occur when serving a large number of users.

For details on converting a legacy VisualAge Generator application, see “Appendix B. Transforming TUIs into Web Transactions” on page 127.

UI record

The *UI record* is a development-time structure that defines a default web page. Generation produces the following:

- An output that can be customized to create a web page of maximum impact and usability.
- A set of outputs that make runtime data available to the customized web page.

At runtime, the web transaction presents the customized web page in either of two ways:

- By referencing the UI record as the I/O object of a CONVERSE.
- By referencing the UI record in an XFER statement.

The UI record is composed of data items, each of which is of a data type (like CHAR or MIXED) and a UI type. For details on the UI record, see Table 2 on page 19.

When development of a UI record is complete, the web designer can act independently of the programmers who work on the business logic. For ideas on how to merge the efforts of web designers and other programmers, see *Building Enterprise Web Transactions using VisualAge Generator JavaBeans and JSPs*, which is cited in the Preface.

Runtime architecture of VisualAge Generator web transactions

VisualAge Generator web transactions operate across three logical tiers (Figure 1).

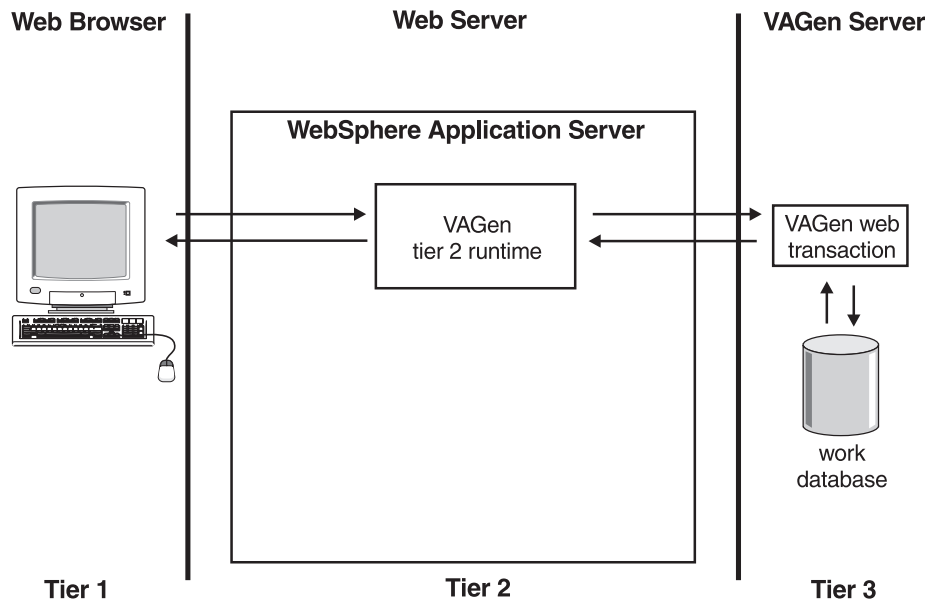


Figure 1. Logical tiers at runtime

Tier 1 is a web browser, where a user types a URL or clicks a link to invoke a program. Tier 2 is a web application server such as WebSphere Application Server, which acts as an intermediary between the browser and the program being invoked. Tier 3 is a server platform, where a generated program accepts data from tier 2, interacts with data stores, manipulates the data, and returns data to tier 2.

The three tiers are usually on separate machines, and the functionality of tiers 2 and 3 can be split across multiple machines.

Figure 2 on page 4 further illustrates the data flow.

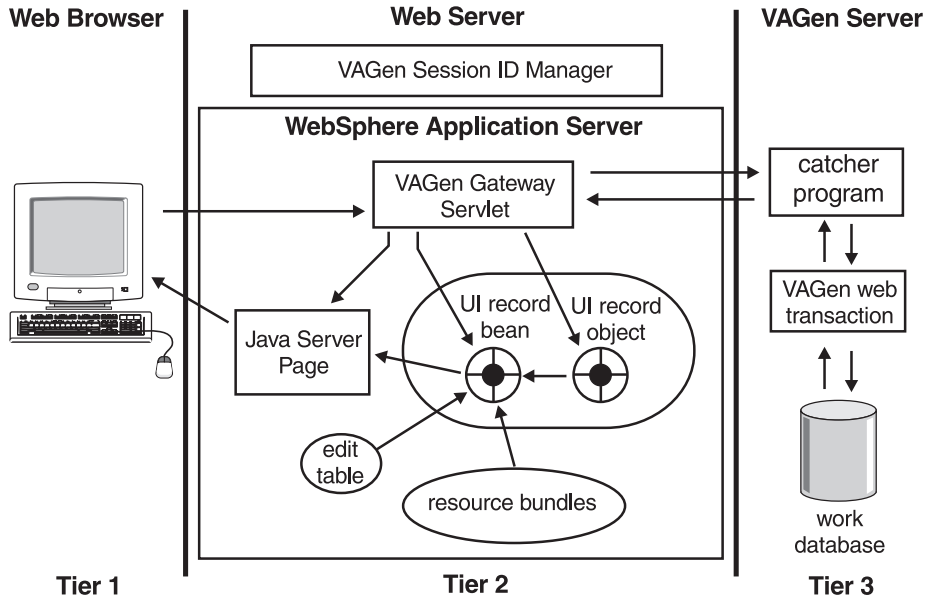


Figure 2. Logical tiers at runtime (detail)

Supported tier 2 environments are AIX, OS/390, OS/400, Solaris, Windows 2000, and Windows NT. Supported tier 3 environments are all those that are available for generated programs other than web transactions, except that MVS TSO, OS/2 CICS, and VM are not supported.

Tier 1

Tier 1 receives a data stream from tier 2, presents that stream to the user, and returns a response based on the user's input. Most of the data received by tier 1 is in Hypertext Markup Language (HTML) format, and the browser renders that stream as a web page.

The HTML received after invoking a generated program consists of one or more HTML <FORM> structures, each of which contains a series of named items. In most cases, the user adds data to those items and clicks a button to submit the data to the same or another program. In other cases, the user clicks a hypertext link to invoke another program, in which case the browser does not send the data provided by the user.

In relation to generated web transactions, a browser session begins when the user accesses the main component on tier 2—the *gateway servlet*. The session ends when the connection is lost, when the browser is closed, or when the user clicks on a logout option provided by the gateway servlet. Timeout options that are set when you configure your web application server also affect when session data is discarded.

Tier 2

At the center of data flow is a series of objects executed in a web application server. Included are Java objects of the following types:

- **Servlet**

A servlet receives a request from a browser and fulfills that request, invoking other objects as necessary. Servlets provide functionality similar to that of Common Gateway Interface (CGI) script but give better runtime performance.

- **Java Server Page**

A *Java Server Page* (JSP) produces an HTML stream and sends the HTML to the appropriate browser. The content of the stream is finalized only at JSP runtime. The web designer who creates a JSP requires only minimal expertise in Java.

- **Java Bean**

A *Java bean* stores runtime data and provides access to that data in accordance with a Java-language specification. In most web-based applications, processing is as follows:

- A servlet uses the set methods of the Java bean to keep data in memory for later use.
- A JSP uses the get methods of the Java bean to embed that data into HTML.

- **Resource Bundle**

A *resource bundle* contains strings to be presented at runtime. An enterprise can make a web transaction clear to a wider audience by translating resource bundles into different human languages.

In relation to VisualAge Generator web transactions, the major components of tier 2 are as follows:

- Installed parts—
 - Gateway servlet
 - Session ID Manager
- Generated parts—
 - UI record JSPs and related objects
 - Edit tables
 - Resource bundles

Gateway servlet

The gateway servlet manages the interaction between the user and a *web application*, which is the set of web-transaction programs available from that gateway servlet. You always assign a separate URL for each gateway servlet.

A gateway servlet does the following:

- Accepts user invocation at a servlet-specific URL.
- Provides a logon page to tier 1, but only the first time that the user invokes the gateway servlet during a given browser session. Use of a logon page is an option that you specify when configuring the gateway servlet.
- Provides an entry page, which gives users access to a list of generated programs. Use of an entry page is also an option when configuring the gateway servlet.
- Accesses the generated programs and returns program-specific data to the browser.

You can configure the gateway servlet to give users access to a single program rather than to an entry page. If an entry page is in use, that page is redisplayed when a generated program terminates.

For details on configuring the gateway servlet, see “Configuring a Gateway Servlet” on page 117.

The gateway servlet presents the optional logon and entry pages by invoking JSPs, as described in the next sections. Also described is the error JSP, which the gateway servlet invokes if a web transaction ends with a terminating error.

Logon JSP: When you configure the gateway servlet, you can specify a *logon JSP*, which sends a logon page that prompts for a userid and password when the user accesses the gateway servlet for the first time in a browser session. Use of a logon JSP is meaningful only if some of the generated programs are running in a password-protected environment such as CICS or IMS.

If your installation has some programs in one password-protected environment and some in another, you must configure a different gateway servlet to access the programs in each password-protected environment, unless the passwords are identical.

Userids and passwords are not passed to programs that run on environments that are free of password protection, so any gateway servlet can provide user access to such programs. To avoid displaying a logon page to users who do not need to specify a password, however, it is recommended that, for password-free programs, you configure a gateway servlet that does not provide access to programs running in password-protected environments.

The password returned from the logon page is unavailable to web transactions, but the userid is available in EZE word EZEUSRID. The userid is different from the user connection ID, which is provided by another runtime component; for details, see “Session ID Manager” on page 7.

VisualAge Generator provides a sample logon JSP, which you can customize.

Entry Page JSP: After the user finishes interacting with a logon page, if any, the gateway servlet either presents the output of a specific program or passes control to an *entry page JSP*, which lists the available programs.

If the gateway servlet is configured to access an entry page JSP, each entry shown to the user includes text that the user can click, as well as a hidden name that is returned to the gateway servlet when the user clicks on the text.

The name returned from the browser must be present in a tier 2 configuration file called the `hptLinkageProperties` file. That file tells the gateway servlet where to find the generated programs and how to communicate with them. The `hptLinkageProperties` file provides the equivalent function of the linkage table for VisualAge Generator Client/Server programs and is described in “Configuring a Gateway Servlet” on page 117.

VisualAge Generator provides an entry page JSP, which you can customize. A link for ending the browser session is included in the HTML sent by that JSP. Instead of using the entry page JSP, you can develop a VisualAge Generator web transaction to display a menu of other programs, in which case you would configure the gateway servlet to give users access to that program rather than to an entry page.

Error page JSP: If an error occurs during processing of a generated web-transaction program and if the error causes the program to end, the gateway servlet invokes the error page JSP, `CSOERRORUIR.jsp`, which presents an HTML stream that tells the user about the problem.

VisualAge Generator provides a default error page JSP to be customized.

Session ID Manager

The Session ID Manager is a VisualAge Generator component that generates two types of unique IDs:

- A *user connection ID*, which is specific to the browser session, is stored in EZE word `EZEUSR`, and is different from the user ID that the user types in a logon page.
- One or more *conversation IDs*, each specific to the browser’s invocation of a generated web transaction. The conversation ID for a given browser and web transaction is stored in `EZELTERM`.

A Session ID Manager may run on a separate machine from the web application server. Moreover, a Session ID Manager may provide IDs to multiple web application servers, each of which may access the same web transactions on tier 3. The IDs generated by a Session ID Manager are unique across all gateway servlets that use it.

The primary purpose of the user connection id and conversation ID is to synchronize information between tiers 2 and 3. One use of the conversation ID is as a key for saving program state in a work database.

UI record JSP and related objects

In addition to the logon, entry page, and error JSPs, the gateway servlet interacts with the following objects to present the generated content of a UI record:

- The *UI record object* (sometimes called the *data bean*) is a Java class instance and is the tier 2 equivalent of the UI record maintained on tier 3.
- The *UI record bean* (sometimes called the *interface bean*) provides data validation and other services on tier 2 and accesses the data contained in the UI record object.
- The UI record JSP does the following—
 - Produces an HTML stream that is based on the characteristics of a UI record.
 - Embeds runtime values in the HTML by invoking methods in the UI record bean.

For example, in the following line from a UI record JSP, the label of field LAST_NAME is placed within the HTML `<b>` structure, which directs the browser to display the string in boldface:

```
<b><%=LAST_NAME.getLabel()%></b>
```

For details on customizing the UI record JSP, see “Chapter 5. Java Server Pages and the UI Record Bean API” on page 63.

When a web transaction executes a CONVERSE or XFER, the following occurs:

1. The generated program sends the UI record data to the gateway servlet.
2. The gateway servlet creates or updates the UI record object, creates the UI record bean (if it does not exist), and invokes the UI record JSP.
3. When producing HTML, the UI record JSP repeatedly calls the UI record bean, which accesses the UI record object so that data from the UI record becomes part of the HTML. Output data formatting is performed on tier 2 by the UI record bean, which formats each output item in accordance with the developer’s specification and only then provides the item to the UI record JSP.
4. The UI record JSP submits the HTML to the browser.
5. After reviewing and possibly changing data on the web page, the user usually clicks a SUBMIT button to send data back to the gateway servlet.
6. On receiving the UI record, the gateway servlet uses the UI record bean to check for input errors, as described in “How Input Data is Edited” on page 16.

Edit Tables

The developer can cause user input to be compared with a range of valid values or with a list of valid or invalid values. Each edit table that contains comparison values is a binary file on tier 2, and each file has a Java wrapper that gives the UI record bean access to the table data.

For details on editing, see “How Input Data is Edited” on page 16.

Resource Bundles

Resource bundles that support generated web transactions are of two types:

- A *UI record resource bundle* includes the title, labels, and help text from a UI record.
- A *message resource bundle* includes the set of messages generated for a table in which you placed those messages. That table is of type MESSAGE or UNSPECIFIED.

At runtime, the following occurs:

1. As a result of a browser setting, the user who invokes a web transaction specifies a *Java locale*, which is a code identifying a human language.
2. The gateway servlet makes the Java locale available to the UI record bean, which loads the appropriate resource bundles.

When the UI record JSP requests a string, the UI record bean accesses the appropriate resource bundle.

For further details, see “Chapter 3. Resource bundles” on page 45.

Tier 3

The user-generated program on tier 3 is of type *Web Transaction*. As necessary, the generated program maintains the state of a user conversation by storing user-specific information in a work database.

Note: Web transactions that run on native AIX, Windows 2000, Windows NT, and Solaris are not segmented and do not use a work database. Web transactions that run on CICS environments use a temporary storage queue as the work database. For details on maintaining the work database on other environments, see “Maintaining the work database on tier 3” on page 115.

The data transmitted between the generated program and the gateway servlet is through a third component, the *catcher*, which resides on tier 3. The catcher invokes the generated program and makes subsequent communication possible. For non-CICS programs, the catcher is a TCP/IP listener; for CICS programs, the catcher is transaction CPMT or an organization-specific equivalent.

An administrator specifies information on the catcher when configuring the gateway servlet, as described in “Configuring a Gateway Servlet” on page 117.

CONVERSE and XFER

The next sections describe how a web transaction uses CONVERSE and XFER, which are the only statements that send data to the browser.

In every tier 3 environment, VisualAge Generator commits all resources and releases all locks when either statement is executed. This behavior prevents a large number of users from locking databases and other resources during *user think time*—the period between the browser’s display of a web page and the user’s submission of the web page to the gateway servlet.

Note: A web transaction can DXFR to another generated program, which itself may be a web transaction. The second program may DXFR to a third generated program, and so on. After each DXFR, the program that contained the DXFR ends, but the automatic commitment of resources occurs only when the final program ends or when one of the programs runs a CONVERSE or XFER.

CONVERSE

When executing a CONVERSE with a UI record as the I/O object, the VisualAge Generator runtime saves the complete program state. An example is illustrated in Figure 3 on page 11.

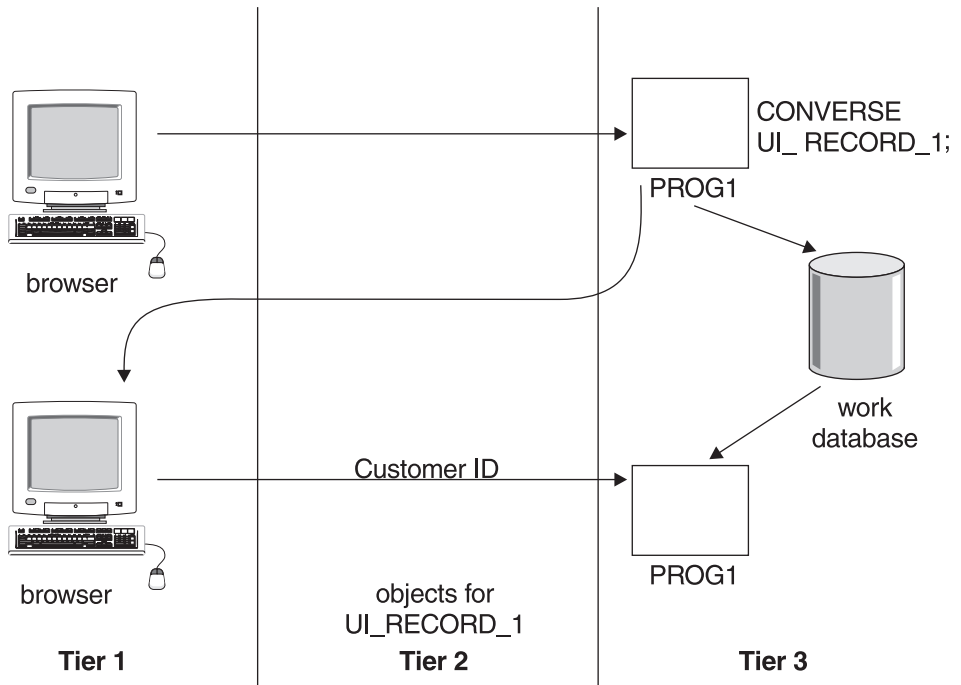


Figure 3. Example of CONVERSE

Processing is as follows:

1. Web transaction *PROG1* executes a CONVERSE for *UI_RECORD_1*, a UI record that prompts for a customer ID. On tier 3, the web transaction saves program state in a work database, ends processing, and transmits data to tier 2. On tier 2, the gateway servlet creates a set of objects for use by the UI record JSP, which sends an HTML stream to the user's browser.
2. The user types a customer ID and submits the page.
3. On tier 2, the gateway servlet uses the existing objects to validate the input and sends a copy of the customer ID to tier 3.
4. On tier 3, the catcher program reinvokes *PROG1*, which restores data from the work database.
5. The web transaction uses the saved data to determine where in the code to continue processing.

When the user clicks a SUBMIT button on a page that was presented by a CONVERSE, an HTML <FORM> structure directs processing to the gateway servlet. The gateway servlet in turn uses a set of hidden fields stored on the HTML form to identify both the UI record objects on tier 2 and the appropriate program on tier 3.

Browser behavior after a web page is presented by a CONVERSE includes the following:

- If the user submits the page, moves to another page in the same browser window, and attempts to return to the page that was presented by the CONVERSE, the browser displays only the entry page or startup program provided by the gateway servlet. The implication for developers is that VisualAge Generator enforces the logical flow of a program when web pages are presented by way of a CONVERSE.
- If the user fails to submit the page, moves to another page in the same browser window, and attempts to return to the page that was displayed by the CONVERSE, the browser is unable to redisplay that page until the user invokes the browser-specific reload feature. A reloaded page includes all output data, but does not include data that the user added.

The page displayed to the user may include SUBMIT buttons or hypertext links that provide access to other programs, but by selecting a checkbox at definition time you can ensure that the returned data is presented in a separate browser window so that the CONVERSE is unaffected.

XFER with a named program

An *XFER with a named program* is a statement that displays a UI record to a user. By submitting the page, the user causes invocation of the named program.

XFER with a named program fulfills the controlled-state model because you determine what state data to save. An example is illustrated in Figure 4 on page 13.

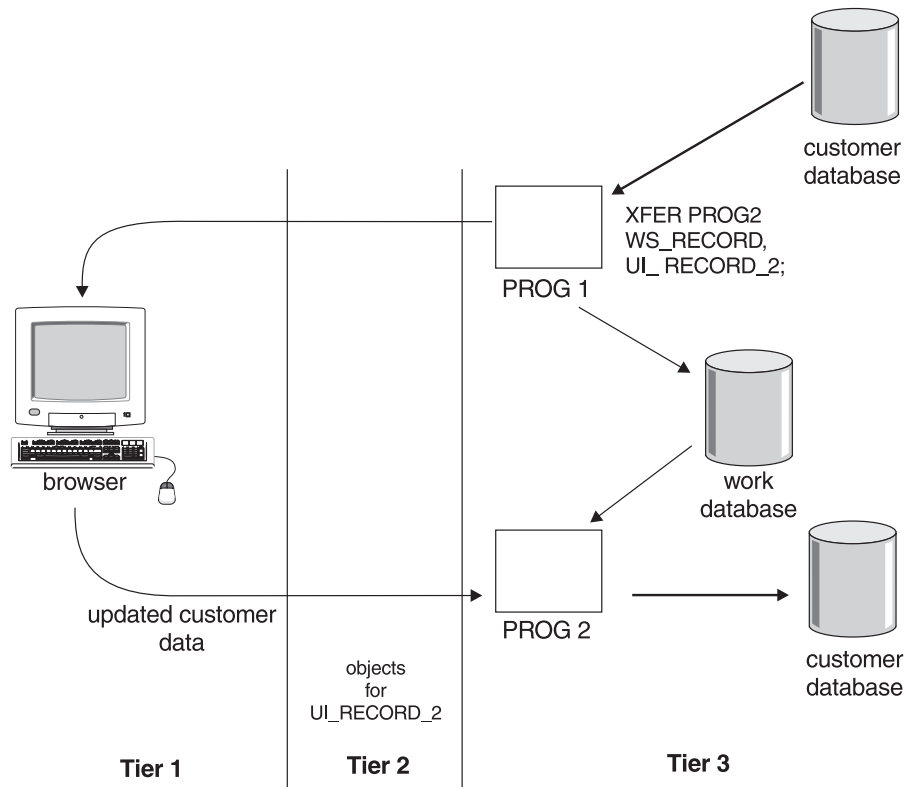


Figure 4. Example of XFER with a Named Program

In the example from the previous section, PROG1 received control after the CONVERSE. The example continues:

1. PROG1 does the following—
 - a. Retrieves database information on the customer specified by the user.
 - b. Places business data in *UI_RECORD_2*.
 - c. Places a timestamp retrieved from the database into working storage record *WS_RECORD*; the timestamp states when the record was last updated.
2. PROG1 executes an XFER with a named program—
`XFER PROG2 WS_RECORD, UI_RECORD_2;`

On tier 3, VisualAge Generator saves the working storage record in the work database, ends PROG1, and transmits data to tier 2. On tier 2, the gateway servlet invalidates the set of objects for PROG1 (making them available for removal by the web server) and creates a set of objects for use by the *UI_RECORD_2* JSP, which sends an HTML stream to tier 1.

3. The user changes the customer data and submits the page.
4. On tier 2, the gateway servlet uses the existing objects to validate the input and sends a copy of the data to tier 3.
5. On tier 3, the catcher program invokes PROG2, which restores the saved working storage record, reads the database record for the customer ID returned in the UI record, checks whether the database record was updated at a time subsequent to the timestamp value in the working storage record, and (if appropriate) writes and commits changes to the database, including a timestamp that reflects the new time of update.

An XFER with a named program is like a CONVERSE across two programs, with the transferred working storage record and UI record providing continuity. Data in working storage remains on tier 3, while data in the UI record is available on tiers 1, 2, and 3.

When the user clicks a SUBMIT button on a page that was displayed by way of an XFER with a named program, an HTML <FORM> structure directs processing to the gateway servlet. The gateway servlet uses a set of hidden fields stored on the HTML form to identify both the UI record objects on tier 2 and the appropriate program on tier 3.

After a page is displayed by way of XFER with a named program, browser behavior includes the following:

- If the user submits the page, moves to another page in the same browser window, and attempts to return to the page that was presented by the XFER, the browser displays only the entry page or startup program provided by the gateway servlet. The implication for developers is that VisualAge Generator enforces the logical flow of a program when web pages are presented by way of an XFER with a named program.
- If the user fails to submit the page, moves to another page in the same browser window, and attempts to return to the page that was displayed by the XFER, the browser is unable to redisplay that page until the user invokes the browser-specific reload feature. A reloaded page includes all output data, but does not include data that the user added.

The page displayed to the user may include SUBMIT buttons or hypertext links that provide access to other programs, but by selecting a checkbox at definition time you can ensure that the returned data is presented in a separate browser window so that the original conversation is unaffected.

XFER with a blank

An *XFER with a blank* is an I/O statement that displays a UI record to a user, retains no state data on tier 2 or 3, and enforces no subsequent processing. On reviewing the web page, the user may access other generated web transactions or—depending on customization of the UI record JSP—may

access web addresses outside of the VisualAge Generator runtime environment. Output from subsequent links may be displayed to the same or a different browser window.

An example is illustrated in Figure 4 on page 13.

In the example described earlier, PROG2 committed changes to the customer database. The example continues (Figure 5):

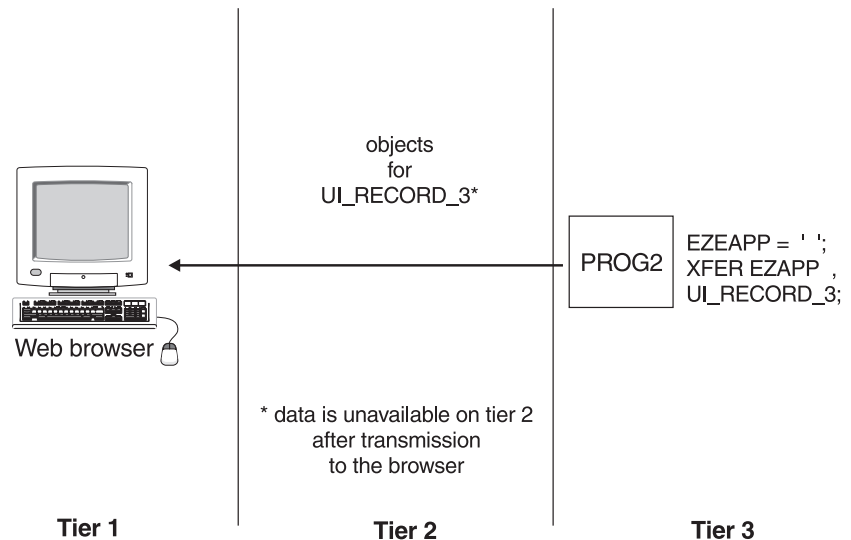


Figure 5. Example of XFER with a Blank

1. PROG2 moves the message "Update Successful!" to a field in UI_RECORD_3.
2. PROG2 executes an XFER with a blank:

```
MOVE ' ' TO EZEAPP;
XFER EZEAPP, UI_RECORD_3;
```

On tier 3, if the code had not committed changes to the customer database, VisualAge Generator would now do so. PROG2 ends, and VisualAge Generator transmits data to tier 2. On tier 2, the gateway servlet uses the set of UI_RECORD_3 objects to send an HTML stream to tier 1 and then invalidates those objects, making them available for removal by the web server.

3. The user sees the web page, which includes a confirmation message but is otherwise unchanged from when the user submitted the form.

4. The user can click the Back button to review the entry page or startup program that is provided by the gateway servlet. Alternatively, the user can link to any program or web address for which access is provided on the web page itself.

For details on what data is transferred when the user clicks a button to access another program, see “Web-page access to other generated programs” on page 24.

When the user clicks the browser’s Back or Forward button to move to a page that was displayed by an XFER with a blank, the browser displays the page, including any input added by the user.

How Input Data is Edited

When defining the UI record, the developer specifies the following:

- The elementary edits (such as numeric formatting) for individual input fields. Any such edits occur on tier 2.
- The table edits (range, match valid, and match invalid) for individual input fields. Any such edits occur on tier 2.
- The edit functions for individual input fields.
- The edit functions for the record as a whole.
- The tier (2 or 3) where the edit function for a given input field occurs.
- The tier (2 or 3) where the edit function for the record occurs.

VisualAge Generator edits a given data item only if the user changed the content.

When tier 2 receives data from the browser, the gateway servlet requests first that the UI record bean perform all the elementary field edits, even if some fail; second (if the prior edits were successful), all the table edits, even if some fail; and third (if the prior edits were successful), all the field-edit functions that the developer had assigned to tier 2, even if some fail. If no errors are detected during the three levels of field edits and if the developer had assigned the record-edit function to tier 2, the UI record bean executes the record-edit function.

If all input edits succeed on tier 2, the gateway servlet passes control to tier 3. If any of the tier 2 edits fail, however, the following occurs:

- The gateway servlet invokes the UI record JSP.
- The UI record JSP calls the UI record bean, which provides the appropriate messages from a language-specific resource bundle.
- The UI record JSP sends the HTML stream to the browser, and no interaction occurs between tier 2 and tier 3.

Elementary field edits and table edits occur only on tier 2, if at all. The developer can minimize response time by also running, on tier 2, any edit functions that do not need the resources of tier 3.

An edit function that runs on tier 2 can access only the UI record to which the function is related. A function that accesses a different web-transaction record or a database must run on tier 3.

Only after edits succeed on tier 2 is control passed to tier 3, where the generated program runs field-edit functions that the developer had assigned to tier 3. If no errors are detected during those edits and if the developer had assigned the record-edit function to tier 3, the generated program executes the record-edit function.

If all input edits succeed on tier 3, the generated program processes the data as expected. If any of the tier 3 edits fail, however, the following occurs:

- The generated program returns message keys and user-defined text inserts to the gateway servlet, which makes that information available to the UI record bean.
- The gateway servlet invokes the UI record JSP, which calls the UI record bean to receive the appropriate messages from a language-specific resource bundle.
- The UI record JSP builds an HTML stream and sends it to the browser.

The behavior of the default UI record JSP is that, if multiple items are in error, an error message is displayed for each.

If the user clicks a button to invoke a program, the gateway servlet brings into memory the objects related to the first UI record in the invoked program. If errors are found, the web page displayed to the user is the web page that is based on that first UI record, not the web page that had been displayed when the user clicked the button.

For additional details on editing, see “Data item edits” on page 43 and “User-defined messages” on page 46.

Chapter 2. UI Types

This chapter describes the UI types, as well as the relationship between the UI types and the HTML produced by the generated UI record JSP. The HTML produced is similar to what is produced by the Integrated Test Facility, which simulates JSP output at test time.

Note: For a quick reference on UI records, see the *VisualAge Generator Programmer's Reference*.

The UI record can include any UI type listed in Table 2.

Table 2. UI Types

UI Type	Description
Form	An item of UI type Form is translated to an HTML <FORM>structure, which is separate from the default HTML <FORM>structure. For details, see “Basic HTML Structure” on page 21. When customizing a UI item of type Form, you specify a program that will be invoked if the user clicks a SUBMIT button from within the derived <FORM>structure. The data submitted to that program can include data from the user as well as data received from the program that presented the web page. For details, see “Web-page access to other generated programs” on page 24.
Hidden	An item of UI type Hidden is translated to one or more HTML <INPUT> tags of type HIDDEN. The value that you assign programmatically to a UI item of type Hidden is available to any program invoked from the web page, but is visible to the user only if the user reviews the web-page source by way of browser-specific keystrokes. If you want to prevent the user from seeing a value, do not assign it to a UI item of type Hidden. Instead, consider using a UI item of type None, as described in “UI type None” on page 40.
Input	An item of UI type Input is translated to an HTML item or items to which the user can assign values. The type of HTML items varies according to other definition-time settings, as described in “HTML from UI type Input or Input/Output” on page 28. Although an item of UI type Input can display values assigned programmatically, VisualAge Generator does not perform output formatting on the item.

Table 2. UI Types (continued)

UI Type	Description
Input/Output	An item of UI type Input/Output is translated to an HTML item or items that have initial display values, which the user can change. The type of HTML items vary according to other definition-time settings, as described in “HTML from UI type Input or Input/Output” on page 28. VisualAge Generator performs both input edits and output formatting on items of type Input/Output.
None	An item of UI type None is translated to a data item that in most cases is excluded from the HTML sent to the browser. For details, see “UI type None” on page 40.
Output	An item of UI type Output is translated to the value of the data item itself. The type of HTML output varies according to other definition-time settings, as described in “HTML from UI type Output” on page 34. The user can not type data to change an Output item but in some cases can select one or more Output items from a list, in which case the selections are made available to the program. VisualAge Generator performs output formatting on items of type Output.
Program Link	An item of UI type Program Link is translated to an HTML <A> tag, which is displayed as a hypertext link. When customizing such an item, you specify a program that will be invoked if the user clicks the hypertext link. The data submitted to that program can include only the data received from the program that presented the web page. For details, see “Web-page access to other generated programs” on page 24.
Submit	An item of UI type Submit is translated to an HTML <INPUT> tag of type SUBMIT. If the user clicks the SUBMIT button that results from that tag, VisualAge Generator on tier 2 performs edits on the user’s input and (if the edits succeed) sends the user data (including the SUBMIT button value) to the program on tier 3. The button value is stored in the submit value item, as described in “Control Items” on page 27. For further details on UI type Submit, see “UI types Submit and Submit Bypass” on page 40.

Table 2. UI Types (continued)

UI Type	Description
Submit Bypass	An item of UI type Submit Bypass is translated to an HTML <code><INPUT></code> tag of type SUBMIT. If the user clicks the SUBMIT button that results from that tag, the button value is stored in the submit value item (as described in “Control Items” on page 27) and is made available to the program on tier 3. The rest of the user’s input is ignored. The primary use of an item of UI type Submit Bypass is to define an Exit button. For further details, see “UI types Submit and Submit Bypass” on page 40.

Basic HTML Structure

The basic structure of a web page is as follows:

```
<HTML>
<HEAD>
<TITLE>
</HEAD>
<BODY>
</BODY>
</HTML>
```

The title you assign to the UI record as a whole is placed in the `<TITLE>` tag. If the title is *My Web Page*, for example, the HTML includes the following:

```
<HEAD>
<TITLE>My Web Page</TITLE>
</HEAD>
```

The generated HTML includes a default `<FORM>` structure, and an additional `<FORM>` structure is present for each UI item of type Form. If, for example, you define one item of UI type Form, the derived HTML `<BODY>` is structured as follows:

```
<BODY>
  <FORM>
  </FORM>
<FORM>
</FORM>
</BODY>
```

Figure 6 on page 22 displays a UI record that is translated to two `<FORM>` structures.

Name	Occurs	Type	Length	Decimals	UI Type	UI Properties	Usage	Description
INSTRUCTION	1	Char	3	0	Output	Custom	Nonshared	
FIRST_NAME	1	Char	20	0	Input/Output	Custom	Nonshared	
LAST_NAME	1	Char	20	0	Input/Output	Custom	Nonshared	
SUBMIT_BUTTON	1	Char	10	0	Submit	Custom	Nonshared	
THE_SECOND_FORM	1	Char	53	0	Form	Custom	Nonshared	
INSTRUCTION2	1	Char	3	0	Output	Custom	Nonshared	
FIRST_NAME2	1	Char	20	0	Input/Output	Custom	Nonshared	
LAST_NAME2	1	Char	20	0	Input/Output	Custom	Nonshared	
SUBMIT_BUTTON2	1	Char	10	0	Submit	Custom	Nonshared	

Figure 6. UI_RECORD_A

Figure 7 displays the derived web page, which reflects values that a program assigned to the UI record items. Some of the HTML that presents the web page is shown later in this section.

My Web Page

Identify an artist:

First Name

Last Name

Identify another artist:

First Name

Last Name

Figure 7. Web page derived from UI_RECORD_A

The first four items in UI_RECORD_A (Figure 6) are translated to tags in a default <FORM> structure. In general, the tags of a default <FORM> structure are derived from UI items that are *not* subordinate to a data item of UI type Form.

The UI items subordinate to THE_SECOND_FORM are translated to tags in a non-default <FORM> structure. In general, the tags of a non-default <FORM> structure are derived from UI items that *are* subordinate to a data item of UI type Form.

You use the two types of <FORM> structures for different purposes, as described in the next sections.

Default <FORM> structure

Following is the HTML derived from the default <FORM> structure described earlier in this chapter:

```
<FORM METHOD="POST" ACTION="http://localhost:8080/hptGateway">
<b>Identify an artist:</b>
<br>
<b>First Name<b>
<INPUT TYPE=TEXT NAME="FIRST_NAME" SIZE=20 MAXLENGTH=20 VALUE="SALVADOR">
<br>
<b>Last Name<b>
<INPUT TYPE=TEXT NAME="LAST_NAME" SIZE=20 MAXLENGTH=20 VALUE="DALI">
<br>
<INPUT TYPE=SUBMIT NAME="SUBMIT_BUTTON" VALUE="Submit the Default Form">

<INPUT TYPE=HIDDEN NAME="hptPageId" VALUE="1381165541">
<INPUT TYPE=HIDDEN NAME="hptAppId" VALUE="PROG1">
<INPUT TYPE=HIDDEN NAME="hptHandlerId" VALUE="12901">
</FORM>
```

In this example, a UI record developer specified the details of the business-related <INPUT> tags—the tags for FIRST_NAME, LAST_NAME, and SUBMIT_BUTTON. However, several other tags—the <FORM> tag and the three <INPUT> tags of type <HIDDEN>—are part of a mechanism by which VisualAge Generator directs processing from the web page to a pre-set program.

In the case of a CONVERSE, the default <FORM> structure directs processing to the program that displayed the web page; and in the case of an XFER with a named program, the structure directs processing to the program named in the XFER statement.

In the case of an XFER with a blank, however, the default <FORM> structure is largely meaningless because it directs processing to tier 2 objects that are no longer in memory. If the user clicks on the SUBMIT button in this case, the gateway servlet displays the entry page or the initial program (depending on the gateway servlet configuration), but does not transfer data from the web page to tier 3.

The web designer who customizes a UI record JSP that produces a default <FORM> structure must avoid modifying the <HIDDEN> tags used by VisualAge Generator. Those tags are preceded in the JSP by the warning "Do Not Modify."

To allow the user to direct processing to a program other than the program identified in a default <FORM> structure, include one of the following in the UI record:

- A data item of UI type Form, along with child data items that include at least one item of type Submit; or
- A data item of type Program Link.

Non-default <FORM> structure

Following is the HTML derived from the non-default <FORM> structure described earlier in this chapter:

```
<FORM METHOD="POST" ACTION="http://localhost:8080/hptGateway?hptAppId=PROG1&execute=1">  
<b>Identify another artist:</b>  
<br>  
<b>First Name<b>  
<INPUT TYPE=TEXT NAME="FIRST_NAME2" SIZE=20 MAXLENGTH=20 VALUE="HENRI">  
<br>  
<b>Last Name<b>  
<INPUT TYPE=TEXT NAME="LAST_NAME2" SIZE=20 MAXLENGTH=20 VALUE="MATISSE">  
<br>  
<INPUT TYPE=SUBMIT NAME="SUBMIT_BUTTON2" VALUE="Submit the Second Form">  
</FORM>
```

In this example, a UI record developer specified not only the details of the business-related <INPUT> tags, but also the <FORM> tag, which specifies what program is to receive control if the user clicks the SUBMIT button. To identify the program, the developer specified a program link within the form itself, as is possible when you develop a non-default <FORM> structure but not when you develop a default one. For details on program linkage, see “Web-page access to other generated programs”.

Web-page access to other generated programs

When customizing a UI item of type Form or Program Link, you specify a generated program for the user to invoke. The data available for submission to that program is different for the different UI types.

For a UI item of type Form, you can submit user data, as is to be expected from an input form on a web page. For a UI of type Program Link, however, you can not. In either case, you can submit data that is in the UI record being presented or in a link parameter you specify.

To specify a program linkage for a UI item of type Form or Program Link, do the following in the Record Editor:

1. Click the UI Properties column for the UI item.
2. When the UI Properties dialog box is displayed, click Program Link, which displays the Program Link tab (Figure 8 on page 25).

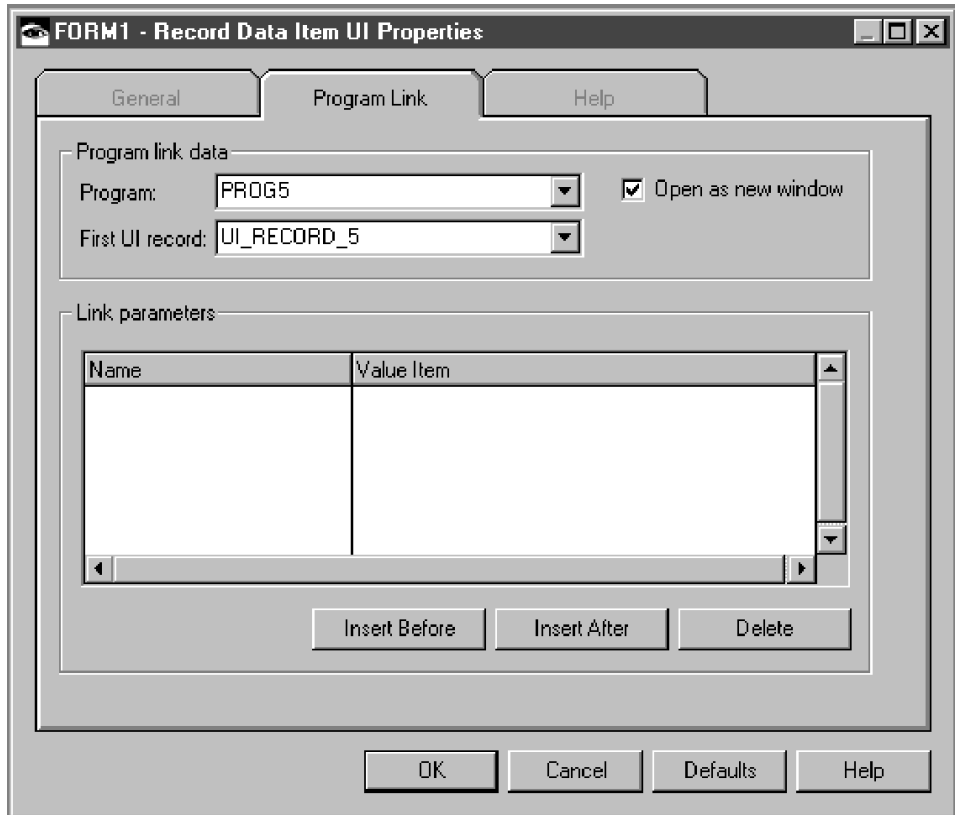


Figure 8. Program Link tab

At the Program Link tab, you identify the program to be invoked, along with its First UI record, if any. You specify the First UI record because otherwise the following would be true:

- The gateway servlet would not access the appropriate record-specific objects for editing input data on tier 2.
- No data would be sent to the program being invoked on tier 3.

The record you specify must be identical in name and structure to the First UI record in the program you are invoking.

To better understand the use of Link Parameters, you need to understand how data is transferred from a non-default <FORM> structure, which is derived from a UI item of type Form, as compared to what occurs when the user clicks a hypertext link, which is derived from a UI item of type Program Link:

- When the user clicks the SUBMIT button on a non-default <FORM> structure, VisualAge Generator copies data from the *submitted form*—the

<FORM> structure that contains the clicked SUBMIT button—to the First UI record. The submitted form may be one of several, but only data from that form is copied automatically to the First UI record.

VisualAge Generator determines what data to copy to the UI record by comparing the names of items in the submitted form to the names of items in the UI record; the copy occurs only when the names are identical.

After copying data in this way from the web page, VisualAge Generator copies any additional values you specify, whether values that were on the web page or constants specified on the Program Link tab at definition time. Values copied from the submitted form reflect any changes made by the user, but values copied from <FORM> structures other than the submitted form are static, reflecting what was present when the web page was displayed.

- When the user clicks a generated hypertext link, the data to be transferred is already in the HTML <A> tag. Consider the following example:
 - The program to be invoked is *PROG5*.
 - The first UI record is *UI_RECORD_5*.
 - The parameters to be passed are *FIRST_NAME* and *LAST_NAME*.
 - The value assigned programmatically to *FIRST_NAME* is *HENRI* and to *LAST_NAME*, *MATISSE*.
 - The label for the UI item of type Program Link is *Continue*.

The derived HTML is as follows:

```
<A HREF="http://localhost:8080/hptGateway?hptAppId=PROG5
&execute=1&record=UI_RECORD_5
&FIRST_NAME=HENRI&LAST_NAME=MATISSE">Continue
```

In this example, user changes to the web page do not affect the data sent to PROG5.

A UI item of type Form or type Program Link can be part of the UI record presented either by a CONVERSE or by an XFER with a named program. In either case, it is recommended that you select the UI property checkbox **Open a New Window** when defining the item of type Form and in that way avoid disrupting the conversation between program and user. A UI item of type Form or Program Link also can be part of the UI record presented by an XFER with a blank, but in this case, use of a secondary window is a matter of preference.

You can link to a URL other than that of the gateway servlet only by customizing the UI record JSP.

Control Items

This section describes the following, optional control items:

- Submit value item
- Occurrences item
- Selected index item

When defining the properties of a UI record as a whole, you can identify a *submit value item*, which is a UI record item that contains the value of the SUBMIT button clicked by the user. You specify, at most, only one submit value item even if the web page contains multiple forms. The submit value item must be of type Char, DBCS, Mixed, or Unicode. If you set no submit value item, the following is true:

- A value for the clicked SUBMIT button is placed in EZE word EZE Aid.
- If the user clicks a SUBMIT button that has a value other than a valid EZE Aid value ('PF1' – 'PF24', 'PA1' – 'PA3', or 'ENTER'), VisualAge Generator interprets the value as being 'ENTER'.

The web designer may convert the SUBMIT button to a clickable image, but doing so has no effect on use of the submit value item. For details on using images in place of buttons, see “How to render an image in place of an HTML SUBMIT button” on page 41.

Note: If the user clicks a button to invoke a program from a non-default form, the gateway servlet brings into memory the objects related to the first UI record in the invoked program. The value of the clicked button is placed into the submit value item of the first UI record in the invoked program, not into the submit value item of the UI record for the web page that had been displayed when the user clicked the button.

When setting the UI properties for an array (a UI item that has an occurs value greater than 1), you can set two other control items:

- The *occurrences item* is a UI record item that contains the number of elements to display on the web page out of all the elements that are in the array. If you set no occurrences item, all the array elements are displayed. The occurrences item must be of type Num, without decimal places or occurs.
- The *selected index item* is a UI record item that indicates the following for elements in an array:
 - Whether an element is set as pre-selected when the web page is displayed.
 - Whether an element is set as selected when data is returned from the web page.

You set the selected index item when setting UI properties for the array. The Selected Index item must be of type Num and must be without decimal places, but may have an occurs greater than 1. If the selected index item has an occurs greater than 1 (if it is an array, although usually an undisplayed one), the following is true:

- When data is being prepared for display, the selected index item contains the indexes of the entries to be set as pre-selected.
- When data is returned, the selected index item contains the indexes of the entries that the user selected.

If the user selected displayed entries with indexes 1, 3, and 5, for example, the first element of the selected index item contains 1, the second element contains 3, the third contains 5, and the rest contain 0.

HTML from UI type Input or Input/Output

The HTML derived from an item of UI type Input or Input/Output depends on several factors, including:

- Whether a Boolean edit check is assigned to the UI item, as is possible when customizing the UI properties.
- Whether the length of the UI item is greater than 80 characters.
- Whether the UI item is an array and, if so, whether the selected index item is present and has an occurs value greater than one.

The effect of a Boolean edit check

Use of a Boolean edit check translates a data item of UI type Input or Input/Output to an HTML <INPUT> structure of type <CHECKBOX>. Figure 9, for example, displays the UI item ACCEPT, which has the label *Good* and the value Y.

Name	Length	Decimals	UI Type	UI Properties
ACCEPT	3	0	Input/Output	Custom

Figure 9. UI item ACCEPT

Figure 10 displays the output as seen by the user.



Figure 10. Checkbox on the web page

The HTML is as follows:

```
<INPUT TYPE=CHECKBOX NAME="ACCEPT" VALUE="Y" CHECKED  
<b>Good</b>>
```

If you assign a Boolean edit check and the data type is Char, a UI item value of *Y* indicates that the item is selected, and a blank indicates that the item is not selected. If the data type is Num, the equivalent values are *1* and *0*. You can use the data-item value in two ways:

- Before the web page is displayed, to define whether the checkbox is pre-selected.
- When analyzing the user's response, to test whether a checkbox was selected.

Other examples of derived HTML in this chapter assume that a Boolean edit check was *not* assigned to the UI item.

The effect of data-item length

An item that is of UI type Input or Input/Output and is defined as 80 characters or less is translated to a one-line textbox on the web page, but a similar UI item defined as more than 80 characters is translated to a multiline textbox. Figure 11, for example, displays the following:

- A 20-character UI item SHORT_STRING, which has the label *Short String* and the value *GOOD JOB!*
- An 81-character UI item LONG_STRING, which has the label *Long String* and the value *MORE SPACE IS NEEDED TO PRESENT AN ITEM THAT HAS A LENGTH GREATER THAN 80.*

Name	Occurs	Type	Length	Decimals	UI Type	UI Properties
SHORT_STRING	1	Char	20	0	Input/Output	Custom
LONG_STRING	1	Char	81	0	Input/Output	Custom

Figure 11. UI items SHORT_STRING and LONG_STRING

Figure 12 on page 30 displays the output as seen by the user.

Short String

Long String

MORE SPACE IS NEEDED TO PRESENT AN ITEM THAT HAS A LENGTH
 GREATER THAN 80.

Figure 12. Long and short input strings on the web page

The HTML is as follows:

```
<b>Short String</b>
<INPUT TYPE=TEXT NAME="SHORT_STRING" SIZE=20 MAX_LENGTH=20 VALUE="GOOD JOB!">
<br>
<b>Long String</b>
<br>
<INPUT TYPE=TEXTAREA NAME="LONG_STRING" COLS=60 ROWS="5" WRAP="virtual">
MORE SPACE IS NEEDED TO PRESENT AN ITEM THAT HAS A LENGTH GREATER THAN 80.</TEXTAREA>
```

The effect of a UI item array

A UI item that is of UI type Input or Input/Output and has an occurs value greater than 1 is translated to an HTML <TABLE> structure. Figure 13, for example, displays a 3-element array of last names.

Name	Occurs	Type	Length	Decimals	UI Type	UI Properties
LAST_NAME	3	Char	20	0	Input/Output	Custom

Figure 13. UI item LAST_NAME, with occurs = 3

Figure 14 on page 31 displays the output as seen by the user, assuming that no selected index item is assigned to the UI item.

Last Name	DALI
Last Name	MATISSE
Last Name	RENOIR

Figure 14. A table of last names on the web page

The HTML is as follows:

```
<TABLE BORDER="0" CELLPADDING="2" CELLSPACING="0">
<TR>
<TD><b>Last Name</b><>
<TD> ALIGN="left"><INPUT TYPE=TEXT NAME="LAST_NAME.1" SIZE=20 MAXLENGTH=20
VALUE="DALI"></TD>
</TR>
<TR>
<TD><b>Last Name</b><>
<TD> ALIGN="left"><INPUT TYPE=TEXT NAME="LAST_NAME.2" SIZE=20 MAXLENGTH=20
VALUE="MATISSE"></TD>
</TR>
<TR>
<TD><b>Last Name</b><>
<TD> ALIGN="left"><INPUT TYPE=TEXT NAME="LAST_NAME.3" SIZE=20 MAXLENGTH=20
VALUE="RENOIR"></TD>
</TR>
</TABLE>
```

The effect of a selected index item (occurs=1)

In relation to a UI item that is an array of type Input or Input/Output, you may assign a selected index item. If the selected index item itself has an occurs value of 1, the user can choose only one value, and the selection column contains radio buttons rather than checkboxes.

When a selected index item called SELINDEX is assigned to the UI item LAST_NAME and has a value of 2, for example, each row in the derived <TABLE> structure includes a radio button, and the second radio button is pre-selected.

Figure 15 on page 32 displays the UI items.

Name	Occurs	Type	Length	Decimals	UI Type	UI Properties
LAST_NAME	3	Char	20	0	Input/Output	Custom
SELINDEX	1	Num	3	0	None	None

Figure 15. UI item LAST_NAME (occurs = 3) and a selected index item (occurs = 1)

Figure 16 displays the output as seen by the user.

☐ **Last Name** DALI

☒ **Last Name** MATISSE

☐ **Last Name** RENOIR

Figure 16. A table of last names on the web page, given a selected index item (occurs = 1)

The HTML is as follows:

```
<TABLE BORDER="0" CELLPADDING="2" CELLSPACING="0">
<TR>
<TD><INPUT TYPE=RADIO NAME="SELINDEX" VALUE="1"></TD>
<TD><b>Last Name</b></TD>
<TD><ALIGN="left"><INPUT TYPE=TEXT NAME="LAST_NAME" SIZE=20 MAXLENGTH=20
VALUE="DALI"></TD>
</TR>
<TR>
<TD><INPUT TYPE=RADIO NAME="SELINDEX" VALUE="2" CHECKED></TD>
<TD><b>Last Name</b></TD>
<TD><ALIGN="left"><INPUT TYPE=TEXT NAME="LAST_NAME" SIZE=20 MAXLENGTH=20
VALUE="MATISSE"></TD>
</TR>
<TR>
<TD><INPUT TYPE=RADIO NAME="SELINDEX" VALUE="2" CHECKED></TD>
<TD><b>Last Name</b></TD>
<TD><ALIGN="left"><INPUT TYPE=TEXT NAME="LAST_NAME" SIZE=20 MAXLENGTH=20
VALUE="RENOIR"></TD>
</TR>
</TABLE>
```

The effect of a selected index item (occurs > 1)

If a selected index item has an occurs value greater than 1, multiple selections from the UI item are possible, and the derived <TABLE> structure includes a checkbox rather than a radio button. For example, if the selected index item

called SELINDEX is itself an array and if the value of SELINDEX element 1 is 1 and the value of element 2 is 3, the first and third checkboxes derived from UI item LAST_NAME are pre-selected.

Figure 17 displays the UI items.

Name	Occurs	Type	Length	Decimals	UI Type	UI Properties
LAST_NAME	3	Char	20	0	Input/Output	Custom
SELINDEX	3	Num	3	0	None	None

Figure 17. UI item LAST_NAME (occurs = 3) and a selected index item (occurs = 3)

Figure 18 displays the output as seen by the user.

☒ **Last Name**

☐ **Last Name**

☒ **Last Name**

Figure 18. A table of last names on the web page, given a selected index item (occurs = 3)

The HTML is as follows:

```
<TABLE BORDER="0" CELLPADDING="2" CELLSPACING="0">
<TR>
<TD><INPUT TYPE=CHECKBOX NAME="SELINDEX" VALUE="1" CHECKED></TD>
<TD><b>Last Name</b></TD>
<TD> ALIGN="left"><INPUT TYPE=TEXT NAME="LAST_NAME" SIZE=20 MAXLENGTH=20
VALUE="DALI"></TD>
</TR>
<TR>
<TD><INPUT TYPE=RADIO NAME="SELINDEX" VALUE="2"></TD>
<TD><b>Last Name</b></TD>
<TD> ALIGN="left"><INPUT TYPE=TEXT NAME="LAST_NAME" SIZE=20 MAXLENGTH=20
VALUE="MATISSE"></TD>
</TR>
<TR>
<TD><INPUT TYPE=CHECKBOX NAME="SELINDEX" VALUE="3" CHECKED></TD>
<TD><b>Last Name</b></TD>
<TD> ALIGN="left"><INPUT TYPE=TEXT NAME="LAST_NAME" SIZE=20 MAXLENGTH=20
VALUE="RENOIR"></TD>
</TR>
</TABLE>
```

The effect of a match-valid table

If you apply a match-valid table as the edit routine for a UI item of type Input or Input/Output, the values in that table are displayed in a list box (an HTML <SELECT> structure), from which the user can select a value.

HTML from UI type Output

The HTML derived from a UI item of type Output depends on several factors, including:

- Whether the UI item is an array.
- Whether the selected index item is present and has an occurs value greater than one.
- Whether the UI item is substructured.

An item of UI type Output

Each item of UI type Output (where occurs = 1) is translated to a pair of strings—the item label, which is displayed in boldface, and the item value. Figure 19, for example, displays an item that has the label *Last Name* and the value *DALI*.

Name	Occurs	Type	Length	Decimals	UI Type	UI Properties
LAST_NAME	1	Char	20	0	Output	Custom

Figure 19. UI item of type Output, occurs = 1

Figure 20 displays the output as seen by the user.

Last Name DALI

Figure 20. Pair of strings on the web page

The HTML is as follows:

```
<b>Last Name</b> Dal i
```

The effect of a UI item array

In the absence of a selected index item, an array of UI items of type Output is translated to a pair of structures—the item label, which is displayed in boldface, and an HTML *preformatted* (<PRE>) structure, which lists the values. Figure 19, for example, displays an item that has the label *Last Name* and the values *DALI*, *MATISSE*, and *RENOIR*.

Name	Occurs	Type	Length	Decimals	UI Type	UI Properties
LAST_NAME	3	Char	20	0	Output	Custom

Figure 21. Array of UI items of type Output

Figure 22 displays the output as seen by the user.

Last Name

```
DALI
MATISSE
RENOIR
```

Figure 22. `<pre>` structure on the web page

The HTML is as follows:

```
<b>Last Name</b><pre>DALI
MATISSE
RENOIR
</pre>
```

Note: The behavior just described is true only if the array of UI items of type Output is not subordinate to a UI item of type Form; that is, only if the values in the array are part of a default `<FORM>` structure. If the array is subordinate to a UI item of type Form, the names in the array are displayed in a list box, as if a selected index item were present. This possibility is illustrated in the next section.

The effect of a selected index item (occurs = 1)

In relation to an array of UI items of type Output, you may assign a selected index item. If the selected index item has an occurs value of 1, the UI item is translated to a pair of structures—the item label, which is displayed in boldface, and an HTML `<SELECT>` structure, from which the user can choose *one* value.

If a selected index item called SELINDEX is assigned to the UI item LAST_NAME and has a value of 2, for example, the browser displays a drop-down list, and the second entry is pre-selected.

Figure 15 on page 32 displays the UI items.

Name	Occurs	Type	Length	Decimals	UI Type	UI Properties
LAST_NAME	3	Char	20	0	Output	Custom
SELINDEX	1	Num	3	0	None	None

Figure 23. Array of UI items, with a selected index item (occurs = 1)

Figure 24 displays the output as seen by the user.



Figure 24. <SELECT> structure on the web page

The HTML is as follows:

```
<b>Last Name</b>
<SELECT NAME="SELINDEX" SIZE="1">
<OPTION VALUE="1">DALI
<OPTION SELECTED VALUE="2">MATISSE
<OPTION VALUE="3">RENOIR
</SELECT>
```

The effect of a selected index item (occurs > 1)

In relation to an array of UI items of type Output, if the selected index item has an occurs value greater than 1, the UI item is translated to a pair of structures—the item label, which is displayed in boldface, and an HTML <SELECT> structure, from which the user can choose *one or more* values.

For example, if the selected index item called SELINDEX is itself an array and if the value of SELINDEX element 1 is 1 and the value of element 2 is 3, the browser displays a drop-down list, and the first and third entries are pre-selected.

Figure 25 displays the UI items.

Name	Occurs	Type	Length	Decimals	UI Type	UI Properties
LAST_NAME	3	Char	20	0	Output	Custom
SELINDEX	3	Num	3	0	None	None

Figure 25. Array of UI items, with a selected index item (occurs >1)

Figure 26 on page 37 displays the output as seen by the user.

Last Name



Figure 26. <SELECT> structure on the web page

The HTML is as follows:

```
<b>Last Name</b>
<SELECT NAME="SELINDEX" SIZE="3" MULTIPLE>
<OPTION SELECTED VALUE="1">DALI
<OPTION VALUE="2">MATISSE
<OPTION SELECTED VALUE="3">RENOIR
</SELECT>
```

The effect of substructuring an array

If the array of items of UI type Output is substructured when a selected index entry is present, the array is translated to an HTML <TABLE> structure. If the selected index item has an occurs value of 1, the user can choose only one value, and the selection column contains radio buttons rather than checkboxes.

Figure 27 displays the UI items when SELINDEX is the selected index item for a substructured array called NAME.

Name	Occurs	Type	Length	Decimals	UI Type	UI Properties
☒ NAME	3	Char	40	0	Output	Custom
☐ LAST_NAME	1	Char	20	0	Output	Custom
☐ FIRST_NAME	1	Char	20	0	Output	Custom
SELINDEX	1	Num	3	0	None	None

Figure 27. Substructured array of UI items, with a selected index item (occurs = 1)

SELINDEX was assigned the value 2; therefore, the second radio button is pre-selected, as shown in Figure 28 on page 38.

Name

	Last Name	First Name
☐	DALI	SALVADOR
☒	MATISSE	HENRI
☐	RENOIR	PIERRE-AUGUSTE

Figure 28. <TABLE> structure on the web page, including radio buttons

The HTML is as follows:

```
<TABLE BORDER="1" CELLPADDING="2" CELLSPACING="0">
<CAPTION ALIGN="top"><b>Name </b></CAPTION>
<TR>
<TH></TH>
<TH>Last Name</TH>
<TH>First Name</TH>
<TR>

<TR>
<TD><INPUT TYPE=RADIO NAME="SELINDEX" VALUE="1"></TD>
<TD ALIGN="left">DALI</TD>
<TD ALIGN="left">SALVADOR</TD>
</TR>

<TR>
<TD><INPUT TYPE=RADIO NAME="SELINDEX" VALUE="2" CHECKED></TD>
<TD ALIGN="left">MATISSE</TD>
<TD ALIGN="left">HENRI</TD>
</TR>

<TR>
<TD><INPUT TYPE=RADIO NAME="SELINDEX" VALUE="3"></TD>
<TD ALIGN="left">RENOIR</TD>
<TD ALIGN="left">PIERRE-AUGUSTE</TD>
</TR>

</TABLE>
```

If the selected index item has an occurs value greater than 1, the user can choose multiple values, and the selection column contains checkboxes.

Figure 29 displays the UI items when SELINDEX is an array.

Name	Occurs	Type	Length	Decimals	UI Type	UI Properties
☒ NAME	3	Char	40	0	Output	Custom
LAST_NAME	1	Char	20	0	Output	Custom
FIRST_NAME	1	Char	20	0	Output	Custom
SELINDEX	3	Num	3	0	None	None

Figure 29. Substructured array of UI items, with a selected index item (occurs > 1)

Elements 1 and 2 SELINDEX were assigned values 1 and 3, respectively; therefore, the first and third checkboxes are pre-selected, as shown in Figure 30.

Name		
	Last Name	First Name
☒	DALI	SALVADOR
☐	MATISSE	HENRI
☒	RENOIR	PIERRE-AUGUSTE

Figure 30. <TABLE> structure on the web page, including checkboxes

The HTML is as follows:

```
<TABLE BORDER="1" CELLPADDING="2" CELLSPACING="0">
<CAPTION ALIGN="top"><b>Name </b></CAPTION>
<TR>
<TH></TH>
<TH>Last Name</TH>
<TH>First Name</TH>
<TR>
<TR>
<TD><INPUT TYPE=CHECKBOX NAME="SELINDEX" VALUE="1" CHECKED></TD>
<TD ALIGN="left">DALI</TD>
<TD ALIGN="left">SALVADOR</TD>
</TR>

<TR>
<TD><INPUT TYPE=CHECKBOX NAME="SELINDEX" VALUE="2" ></TD>
```

```

<TD ALIGN="left">MATISSE</TD>
<TD ALIGN="left">HENRI</TD>
</TR>

<TR>
<TD><INPUT TYPE=CHECKBOX NAME="SELINDEX" VALUE="3" CHECKED></TD>
<TD ALIGN="left">RENOIR</TD>
<TD ALIGN="left">PIERRE-AUGUSTE</TD>
</TR>

</TABLE>

```

UI type None

A data item of UI type None generates a data item that in most cases is excluded from the HTML sent to the browser. The data item is available on tiers 2 and 3.

You can use a data item of type None as any of the control items described in “Control Items” on page 27. In addition, you can use the data item as part of an edit function running on tier 2, but only if the UI record is presented either by an XFER with a named program or by a CONVERSE.

If a UI record is presented by an XFER with a blank, you cannot store a data item of type None on tier 2 for use after the user submits the page, because after the user submits the page, the UI record bean and UI record object are created with data from the browser and only from the browser.

If a data item of UI type None is used as a link parameter in a data item of UI type Submit, Submit Bypass, or Program Link, VisualAge Generator treats the data item of UI type None as being of UI type Hidden and includes the generated data item in the HTML sent to the browser. For an overview of UI types, see Table 2 on page 19.

UI types Submit and Submit Bypass

This section further describes the effect of data items of UI type Submit or Submit Bypass. Topics are as follows:

- How values and labels are translated.
- How to render an image in place of an HTML SUBMIT button.

How values and labels are translated

To display a SUBMIT button, you must specify a data value for an item of UI type Submit or Submit Bypass. Values assigned in the program override values assigned at definition time. To prevent the SUBMIT button from being displayed, assign a blank.

In most cases, if you specify a label, the label is the button text, and if you fail to assign a label, the data value is the button text. One exception is a button defined as a child of a data item that has an occurs greater than one. In this case:

- The button label is translated to be the name of a column in an HTML <TABLE> structure.
- The button value is translated to be both the name displayed on a given button and the button value itself.

To assign a different value to each button in the column, assign different values in the program. To assign the same label to each button in the column, customize the HTML in the UI record JSP; specifically, include a static value for NAME in place of calls to the data-item-specific method `getLabel()`. For details on customizing JSP, see “Chapter 5. Java Server Pages and the UI Record Bean API” on page 63.

How to render an image in place of an HTML SUBMIT button

To substitute an image for a SUBMIT button, customize the HTML in the generated UI record JSP—

- Substitute TYPE=IMAGE for TYPE=SUBMIT.
- Identify the source of the image.

An example HTML tag:

```
<INPUT NAME=REFRESH TYPE=IMAGE SRC=refresh.gif>
```

To trap the image-specific X and Y coordinates that are received when the user clicks on an image that is used in place of a SUBMIT button, add two items with a numeric data type to the UI record. The name of the item that receives a coordinate at runtime corresponds to the HTML NAME tag for the button, but the name of the item in the UI record has an additional X or Y. For instance, the item REFRESHX receives the X coordinate of the image defined in the previous example, and the item REFRESHY receives the Y coordinate.

The X and Y coordinates are relative to the top-left corner of the image and are positive integers that identify a number of pixels.

If you specify an array of items of UI type Submit or Submit Bypass:

- The HTML NAME tag for each element is formatted as follows, where *item_name* is the name of the UI item, and *n* is a one-based index value:
Item_name.n

The name of the first element in an array of images that corresponds to UI item REFRESH, for example, is as follows:

REFRESH.1

- To trap the image-specific X and Y coordinates that are received when the user clicks on one image in an array, use an array of appropriately named numeric items; in this example, REFRESHX and REFRESHY are themselves items that have occurs greater than one. If the user clicks REFRESH.1, the first element of REFRESHX receives an X coordinate that is relative to the top-left corner of that image.

To prevent use of incorrect values, set the X and Y fields for a given SUBMIT button to zero in the lines that precede the CONVERSE or XFER.

The effect of parent UI items

The following is true of HTML output derived from UI items (of type Input, Input/Output, or Output) that are defined with occurs = 1:

- If the UI items lack a parent item, or if the only parent item is of UI type Form, VisualAge Generator places a break tag (
) between the output derived from one UI item and the output derived from the next. As a result, the browser displays the output on different lines.
- If UI data items have the same parent (of a type other than Form), the break tag is missing, and the browser tries to display all the output on the same line.

Help text

Help text is available at runtime, but you must tailor the UI record JSP to access that text from the UI record bean. You may want to include a client-side script in the UI record JSP so the user can easily access the help text.

Web page attribute setting

Although DISABLED and READONLY are attributes of HTML <FORM> structures, you cannot set UI record data items as you can for 3270 map fields.

If you wish to use the DISABLED or READONLY attributes, add a data item as a flag in the UI Record for any field you want to protect or unprotect; then, customize the generated JSP so that the generated HTML is set in accordance with the runtime value of that flag.

Note: When the user submits a form, values from DISABLED fields are *not* sent to tier 2.

If you wish to affect cursor positioning, include a client-side script when customizing the UI record JSP; and to set colors or other extended attributes, use a mixture of client-side script and style sheets.

An IF statement like the following is valid in the web transaction:

```
IF THIS_UI_ITEM IS MODIFIED;
```

You can not set a UI item modified from within the code.

Data item edits

The following edits implement processing similar to that provided for map fields in text user interface programs:

- Input required
- Minimum input (characters)
- Check SO/SI space
- Minimum value
- Maximum value
- Zero edit
- Numeric formatting (separators, decimal point, sign)
- Folding

The following edits work differently in a UI record:

- Currency symbol is the same as in a map definition, but a 1–3 character currency symbol is also supported.
- A date-and-time edit indicates that the following runtime checks occur for the data item:
 - Datetime values that pass between the browser and tier 2 in either direction must correspond to the format that the web designer establishes in the UI record JSP. The web designer establishes a datetime format by invoking the data-item-specific method `setDateTimeFormat()`, as described in “void setDateTimeFormat(*DateFormat_object*);” on page 70. If the web designer establishes no format, the format sent from the browser is used; specifically, the default datetime format of the browser’s Java locale.
 - If the data item is non-numeric, the datetime values that pass between tier 2 and tier 3 in either direction must correspond internally to a long gregorian format for date and time. The format is specified during gateway-servlet configuration when you set parameter `hptDateMask`. For details, see “Configuring a Gateway Servlet” on page 117.

For web transactions generated in C++, the format in `hptDateMask` must match that specified in environment variable `EZERGRGL`. For web transactions generated in COBOL, the format in `hptDateMask` must match the datetime format specified during installation of VisualAge Generator Server.
- Justification is not supported, so fill characters do not work well. Null is not a valid fill character.
- Hex edit is not supported.

New edits for a UI record data item include Boolean, as described in “The effect of a Boolean edit check” on page 28.

Only the lowest layer in a substructure can have edits defined.

Chapter 3. Resource bundles

A resource bundle is a Java object that contains strings to be presented at runtime. The content of a resource bundle generated for a UI record includes the web-page title, labels, and help text; and the content of a resource bundle generated for a message table includes a set of messages. A resource bundle is specific to a human language, and you can make a web transaction clear to a wider audience by translating resource bundles into different languages.

Resource bundles at runtime

At runtime, the following occurs:

1. As a result of a browser setting, the user who invokes a web transaction specifies a *Java locale*, which is a code identifying a human language.
2. The gateway servlet makes the Java locale available to the UI record bean, which loads the appropriate resource bundles.

Each subset of information in the Java locale provided by the browser is separated from the next by an underscore. For example, *en* represents English, and *en_US* represents United States English (as distinct from British English). You could also add a third value as in *no_NO_B*, which specifies a particular variant of Norwegian. The first two items—the *language code* and the uppercase *country code*—are based on a Java language specification; and any additional items constitute a *variant*, which is not part of a Java specification.

If the user requests a Java locale that is precisely matched in the name of an available resource bundle, that resource bundle is the source of strings presented to the user. The user may request a Java locale that is more specific than the locale in the name of the available resource bundle—for example, the user may specify *no_NO_B* when the only the Norwegian-language resource bundle is named with *no_NO*; in this case, the less-specific bundle is the source.

It is good practice to provide a *default resource bundle*, the name of which includes no Java locale. A default resource bundle becomes the source if the user specifies either no locale or a locale that is less specific than the locale in the names of other bundles. If the user specifies *en* when the only locale-specific bundle includes the name *en_US*, for example, the default bundle is accessed.

If no resource bundle can be loaded for a UI record, the string values specified at definition time are presented to the user. If no resource bundle is

available when the web transaction attempts to access a user-defined message, VisualAge Generator presents output from the error-page JSP and removes tier 2 objects to terminate the user conversation.

Resource bundles at generation time

Two generation options control the production of resource bundles:

- GENRESOURCEBUNDLE
- RESOURCEBUNDLELOCALE

GENRESOURCEBUNDLE is necessary in the following cases:

- You are generating only a UI record and want a resource bundle for it.
- You are generating a table used for web-transaction messages.
- You are generating a web transaction and want a resource bundle for each UI record.

When you are generating a web transaction and have identified a table as the message table, VisualAge Generator produces a message resource bundle regardless of whether you set GENRESOURCEBUNDLE.

In any case, if you want to specify a Java locale, set RESOURCEBUNDLELOCALE.

The source of a message resource bundle is a message table, and the definition-time table name is composed of a 1–4 character prefix followed by a 3-character NLS value. At generation time, the NLS value must be matched by the setting of generation option TARGNLS.

User-defined messages

A message in a resource bundle can be referenced by EZE word EZEUIERR, as described in the *VisualAge Generator Programmer's Reference*. In addition, a message can be associated with a data item at data-item definition, such that if the edits for the data item fail, the specified message is displayed on the web page.

A message table used by a web transaction may be of type MESSAGE or UNSPECIFIED and must be defined with at least two columns, each no longer than 254 bytes:

- Column 1 may be of type NUM4, CHAR, or MIXED. The value of column 1 is a key for accessing a specific message. If the key is numeric, leading zeros are prepended to it and the character equivalent is used as a key.
- Column 2 may be of type CHAR or MIXED. The value of column 2 is a message that may include inserts, which are placeholders that receive

values when the message is accessed. The format of the insert is as follows, where *integer* has an integral value in the message table itself:

{*integer*}

An example for column 2:

The value for field {1} is not between {2} and {3}.

At runtime, VisualAge Generator uses the key you specify in EZEUIERR or in a data-item definition to access a specific message. The rules are as follows:

- If the key specified in EZEUIERR or the data-item definition is numeric, VisualAge Generator uses the character equivalent to search for a message.
- If the key specified in EZEUIERR or the data-item definition is non-numeric, VisualAge Generator removes any leading and trailing spaces to search for a message. If no match is found, VisualAge Generator tries again by using the value as specified.
- If a message is not available, VisualAge Generator presents output from the error-page JSP and removes tier 2 objects to terminate the user conversation.

Chapter 4. Testing Web Transactions

When developing web transactions, you can test them in several ways:

- You can test them in the Integrated Test Facility (ITF), which emulates runtime processing. For details, see “ITF”.
- You can integrate generated programs (which run in target environments) with test programs (which run in ITF). The VisualAge Generator facility that makes this possible is *Callable ITF*. For details, see “Callable ITF” on page 50.
- You can customize the UI record JSP and review its output without accessing the generated program on tier 3. The VisualAge Generator facility that makes this possible is *Author-Time Values*. For details, see “Author-Time Values” on page 53.
- If you are using VisualAge for Java, you can test programs in the WebSphere Test Environment (WTE), which is tightly integrated with the development environment and provides a subset of the function provided by the WebSphere Application Server. For details, see “Chapter 4. Testing Web Transactions”.

WTE can be used in conjunction with Callable ITF and Author-Time Values.

ITF

When you are testing a web transaction in ITF and run a command that sends data to the user, ITF does the following:

- If necessary, launches the web browser that is your workstation default.
- Presents a web page to that browser by sending an HTML stream, which approximates the stream that would be produced by the generated UI record JSP.
- Waits for you to do one of the following :
 - Submit a form from the web page; in this case, input and SUBMIT-button data is returned to ITF.
 - Click on a hypertext link; in this case, the only data returned is the value of linkage parameters you specified for the item of UI type Program Link.
- Performs all edits defined for the UI items of type Input. If any edits fail, ITF re-creates the web page and places error messages under each field whose value failed an edit.

When a web transaction exits, ITF displays an entry page that lists all web transactions that are available for testing. This behavior simulates what occurs when a program terminates at runtime.

Callable ITF

Callable ITF provides a way to invoke a web transaction in ITF from a gateway servlet. You use Callable ITF to integrate generated programs (which run in the environment for which they were generated) with test programs (which run in ITF).

Figure 31 illustrates the relationships.

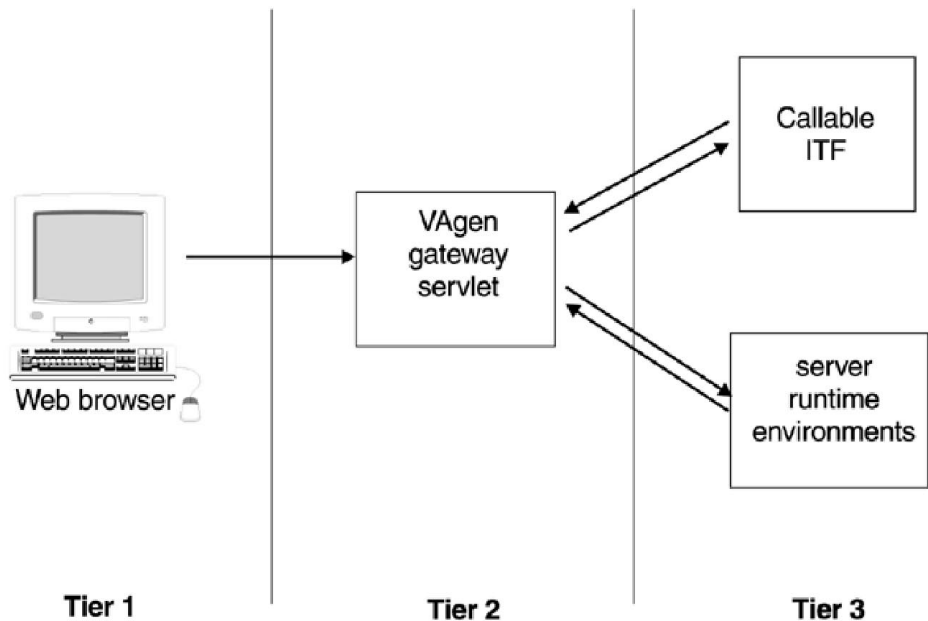


Figure 31. Callable ITF

Although ITF runs on the same workstation as the browser, Callable ITF is itself a server environment that runs as tier 3. Other tier 3 programs may run on different machines, and the gateway servlet sends data to and from each program.

Scenarios include the following:

- You run a generated program that XFERs to a test program.

- You run a generated program that presents a web page which contains buttons or hypertext links that you click to invoke a test program.
- You configure the gateway servlet so it presents an entry page that gives access to one or more test programs.
- You configure the gateway servlet so that it invokes a single test program instead of presenting an entry page.

A test program can XFER to a generated program or to another test program.

Note: A DXFR occurs solely on tier 3, with no involvement by the gateway servlet. Although a runtime program can DXFR to a generated program, and a test program can DXFR to another test program, the following restriction applies: Neither a generated program nor a test program can DXFR to a program of the other kind.

To provide services between the browser and the test program, you need to *generate* the tier 2 outputs for the test program and to provide those outputs to the web application server. (Although a program that runs in Callable ITF could be described as a *generated test program*, the phrase *test program* is used to distinguish that type of program from programs that are executing in a production environment.)

To set up Callable ITF:

- Assign a TCP/IP port to the ITF Listener. For details on the procedure, see “Assigning a TCP/IP port to Callable ITF”. The ITF uses that port to listen for messages from the gateway servlet.
- Configure the web application server, and make the session ID manager and gateway servlet available to that server. For details on configuring the WebSphere Application Server, see “Chapter 7. Running Web Transactions in WebSphere Application Server” on page 111. For details on configuring the WebSphere Test Environment, see “WebSphere Test Environment” on page 54.
- Configure the gateway servlet to provide access to each test and generated program. For details, see “Configuring the gateway servlet” on page 52.
- Generate the test programs (and, if necessary, the generated programs) and deploy the resulting tier 2 objects to the web application server.

Assigning a TCP/IP port to Callable ITF

To assign a TCP/IP port to Callable ITF:

1. At the VAGen Parts Browser, click **Tools>Test Facility Web Listener**.
2. The Web Server Listener dialog box is displayed (Figure 32 on page 52).



Figure 32. Test Facility Web Listener dialog box

Type a number that identifies a TCP/IP port that is not otherwise active on the machine and click **Start Listener**. By this action you have started the tier 3 listener for Callable ITF; when you deploy web transactions in a production environment, you will need to configure and start a listener (or other type of catcher) there as well.

Configuring the gateway servlet

For general details on configuring the gateway servlet, see “Appendix A. Configuring the Gateway Servlet” on page 117. In relation to Callable ITF, use the following identifiers in the file referenced by parameter `hptLinkageProperties`:

- **commtype** is *TCPIP*.
- **contable** identifies a conversion table listed in the *VisualAge Generator Client/Server Communications Guide*. If the machine on which Callable ITF runs uses a code page and binary format for English characters (US or UK), do as follows:
 - If Callable ITF is running on Windows, specify *CSOI1252*.
 - If Callable ITF is running on OS/2, specify *CSOI1437*.
- **location** is the host name for the machine running Callable ITF.
- **serverid** is the TCP/IP port that you specified in the Test Facility Web Listener dialog box, as described in “Assigning a TCP/IP port to Callable ITF” on page 51.
- **javaProperty** is the package that contains the UI record beans for the test program.

An example set of file entries for Callable ITF is as follows:

```
application.PROG1=itf
```

```
serverLinkage.itf.commtype=TCPIP
```

```
serverLinkage.itf.contable=CSOI1252
serverLinkage.itf.location=machine01
serverLinkage.itf.serverid=1401
serverLinkage.itf.javaProperty=my.pkg
```

Identifying and deploying tier 2 objects

For details on deploying objects to either the WebSphere Test Environment or the WebSphere Application Server, see “Deploy web-transaction files to tier 2” on page 60.

Author-Time Values

Author-Time Values is a VisualAge Generator facility that lets the web designer work on the web page without accessing the generated program. Separating the web designer from the complexities of tier 3 gives that person more control of the page-design environment, which can be in a separate location from the development environment.

When a generated web transaction is running in production, the gateway servlet invokes the UI record JSP, which is the source of HTML sent to the browser. When Author-Time Values is in effect, however, a second option is available: The web designer can invoke the UI record JSP directly from a browser by specifying the web address for that JSP, in which case the output web page includes a set of initial values.

The initial values reflect only the data-item characteristics. For example, 99.99 is displayed for an item of UI type Output, data type Num, length 4, decimals 2.

The Author-Time Values facility is available when a program or UI record is generated with option GENAUTHORTIMEVALUES. That option adds the method `initAuthorTimeValues()` to the UI record bean and includes a scriptlet in the UI record JSP to invoke that method.

To produce a realistic set of initial data values, change the UI record bean, but limit your changes to the data values in method `initAuthorTimeValues()`. Assignments to the first three elements of an array of data items of type CHAR, for example, are as follows:

```
UI_RECORD_1.LAST_NAME.assign(0,"AAAAA");
UI_RECORD_1.LAST_NAME.assign(1,"AAAAA");
UI_RECORD_1.LAST_NAME.assign(2,"AAAAA");
```

In the example, change only the characters AAAAA. The first parameter for the `assign` method (number 0, 1, or 2) refers to the array index, which is zero-based in Java. If a data item is not an array, the first parameter is 0.

A second use of Author-Time Values is to emulate runtime-edit error messages. To see what message output looks like, add a statement like the following in `initAuthorTimeValues()`:

```
getLAST_NAME().setErrorMsg( 0, "Value entered for data item LAST_NAME is invalid");
```

No error checking occurs if you access the UI record JSP instead of the gateway servlet, but the message in quote marks is displayed on the web page until you comment out the line that contains the message. Again, the first parameter of the `setErrorMsg` method refers to an array index, if any.

For your convenience, the `setErrorMsg` statement is available within comments for input fields listed in `initAuthorTimeValues()`; you need only to uncomment the statements until you finish displaying the message.

To display the sample output of a UI record JSP, include the JSP name instead of the gateway-servlet name when typing the URL. To access `UI_RECORD_1.JSP`, for example, enter a URL like the following:

```
http://localhost:8080/UI_RECORD_1.JSP
```

Although gateway-servlet invocation of the JSP is still possible when Author-Time Values is in effect, the gateway servlet is not required to be in the web application server. All that is required there is the UI record JSP, UI record bean, and UI record object.

The web page can include hypertext links, but when you access the UI record JSP directly, you can not test interaction between web pages.

WebSphere Test Environment

The WebSphere Test Environment (WTE) is a VisualAge for Java feature that provides a subset of the functionality found in the WebSphere Application Server. This feature lets you group HTML files, servlets, and JSPs into a logical unit called a *web application*. Also, this feature provides a JSP execution monitor and JSP compiler, as well as a servlet engine that runs code such as the VisualAge Generator gateway servlet.

To make WTE available:

1. Install the EJB/JSP Development Environment from the VisualAge for Java installation CD.
2. Load the feature *IBM EJB Development Environment*.

Before interacting with WTE, import the gateway servlet into its own VisualAge project called *IBM VisualAge Generator Gateway*. The servlet class file is usually installed in `C:\Ibmvgagen\Vgwgs45` and is called `hptGateway.jar`.

Configuring the test environment

By the procedures described later in this section, configure the test environment:

1. In the WTE, configure a logical unit called a *web application*, which includes the gateway servlet as well as one or more generated web transactions—
 - a. Define the web application to WTE.
 - b. Configure the web application itself.
2. Include resources in the WTE *classpath*, which is the list of directories from which WTE accesses Java classes.
3. Complete the general set up—
 - a. Configure the gateway servlet.
 - b. Place the gateway-servlet JSPs where WTE can access them.
4. Prepare and deploy the tier 2 files for each web transaction.

Note: The instructions in the next sections assume that VisualAge for Java was installed in directory Program Files\Ibm\VisualAge for Java and that the web application is named `practice_app`.

Define a new web application to WTE

1. Using Windows Explorer, go to the following directory in the drive where you installed VisualAge for Java—
Program Files\Ibm\VisualAge for Java\ide\project resources
IBM WebSphere Test Environment\properties
2. Copy file `default_servlet_engine` to `default_servlet_engine.bak`.
3. Open `default_servlet_engine` in a text editor. A series of XML tags is displayed.
4. Within the structure that begins `<websphere-servlet-host name="default_host">` is a `<websphere-webgroup>` structure, which begins and ends as follows—

```
<websphere-webgroup name="default_app">
</websphere-webgroup>
```

Copy that `<websphere-webgroup>` structure and place it immediately after the original.

5. Customize the new structure as follows—

```
<websphere-webgroup name="practice_app">
<description>practice webgroup</description>
<document-root>c:/webroot1</document-root>
<classpath>c:/webclass1;$approot$/servlets</servlet_path>
<root-uri>practice</root-uri>
<auto-reload enabled="true" polling-interval="3000"/>
<shared-context>false</shared-context>
</websphere-webgroup>
```

The setting of <document-root> specifies that WTE can access HTML files and JSPs in directory c:\webroot1. You must specify a single directory, not a concatenation of several.

The setting of <classpath> specifies that WTE can access Java classes from the specified directory or (as in the example) from a concatenation of directories (in this case, the concatenation includes c:\webclass1). The symbol \$approot\$ refers to the webgroup directory path, which in this case is as follows—

```
Program Files\Ibm\VisualAge for Java\ide\project resources  
\IBM WebSphere Test Environment\hosts\default_hosts\practice_app
```

The setting of <root-uri> specifies that the URL for accessing the webgroup contains /practice immediately after the host name and port number. When you configure the web application itself, you will add a qualifier; for now, the URL is as follows:

```
http://localhost:8080/practice
```

Configure the web application

To configure the web application, you create a subdirectory for that web application, then customize the contents.

To create the subdirectory:

1. Using Windows Explorer, go to the following directory in the drive where you installed VisualAge for Java—

```
Program Files\Ibm\VisualAge for Java\ide\project resources  
\IBM WebSphere Test Environment\hosts\default_hosts
```

2. Copy the subdirectory default_app to a new subdirectory under default_hosts; name the new subdirectory practice_app.

To customize the new subdirectory:

1. Within practice_app\servlets, rename default_app.webapp to an appropriate name; practice_app.webapp, in this case.
2. Open practice_app.webapp in a text editor. A series of XML tags is displayed.
3. Customize the first <name> and <description> tags—
4. Copy the last <servlet> structure in the file and place that copy in front of the original. Customize the new structure as follows—

```
<name>practice_app</name>  
<description>practice application</description>  
  
<servlet>  
<name>GatewayServlet</name>  
<description>VA Generator Web Transaction servlet</description>  
<code>com.ibm.hpt.gateway.GatewayServlet</code>  
<servlet_path>/GatewayServlet</servlet_path>
```

```

<init-parameter>
<name>hptGatewayProperties</name>
<value>C:/linktabs/gw.properties</value>
</init-parameter>
<autostart>true</autostart>
</servlet>

```

The setting of `<servlet_path>` specifies that the URL for accessing the gateway servlet ends in `/GatewayServlet`. The complete URL:
`http://localhost:8080/practice/GatewayServlet`

The setting of `<init-parameter>` specifies that the initialization parameters for the `GatewayServlet` are in `C:\linktabs\gw.properties`. (WTE requires that file paths be identified with forward slashes.)

Note: You may not specify file paths that include spaces, even if you use quote marks to delimit the name (you gain nothing by specifying `"c:\My Files\gw.properties"` instead of `c:\My Files\gw.properties`).

To identify a path that includes spaces, use the file or directory name assigned by DOS for each identifier that has spaces. The name for directory `PERSONAL COMMUNICATIONS`, for example, might be `PERSON~1`. To derive the name for an identifier that has spaces:

- a. Open a DOS command prompt.
- b. Go to the directory that contains the subdirectory or file whose name has spaces.
- c. Type the following command and press **Enter**—
`dir *.* /X`

Subdirectories and files are each listed with the DOS-assigned name, if any, preceding the other name.

5. Save the file.

Include resources in the WTE classpath

To include resources in the WTE classpath:

1. From the VisualAge for Java Workbench, click **Workspace>Tools>WebSphere Test Environment**.
2. The WebSphere Test Environment Control Center dialog box is displayed (Figure 33 on page 58).

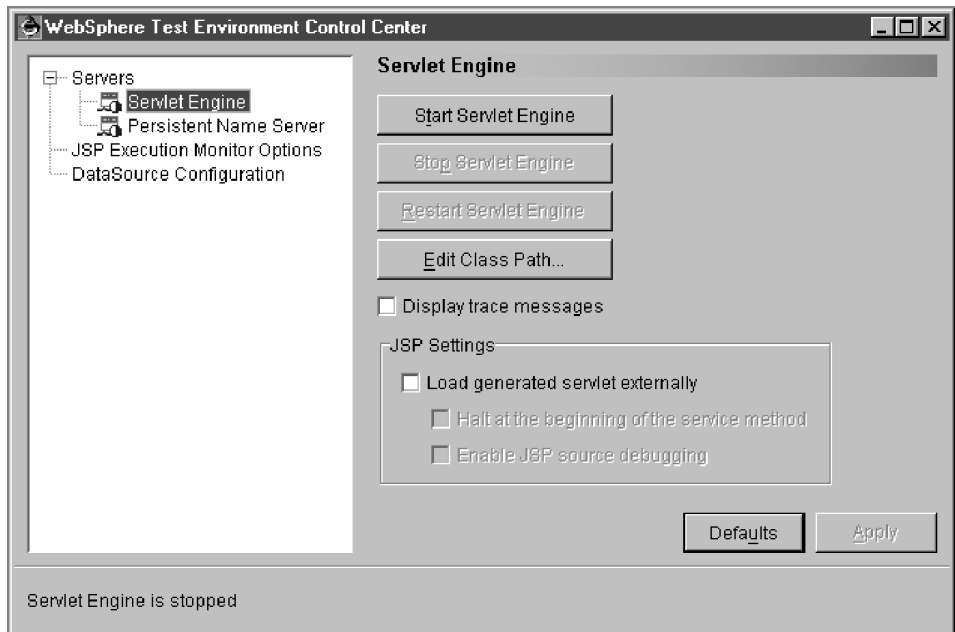


Figure 33. WebSphere Test Environment Control Center dialog box

3. Click **Edit Class Path**.
4. The Servlet Engine Classpath dialog box is displayed (Figure 34 on page 59).

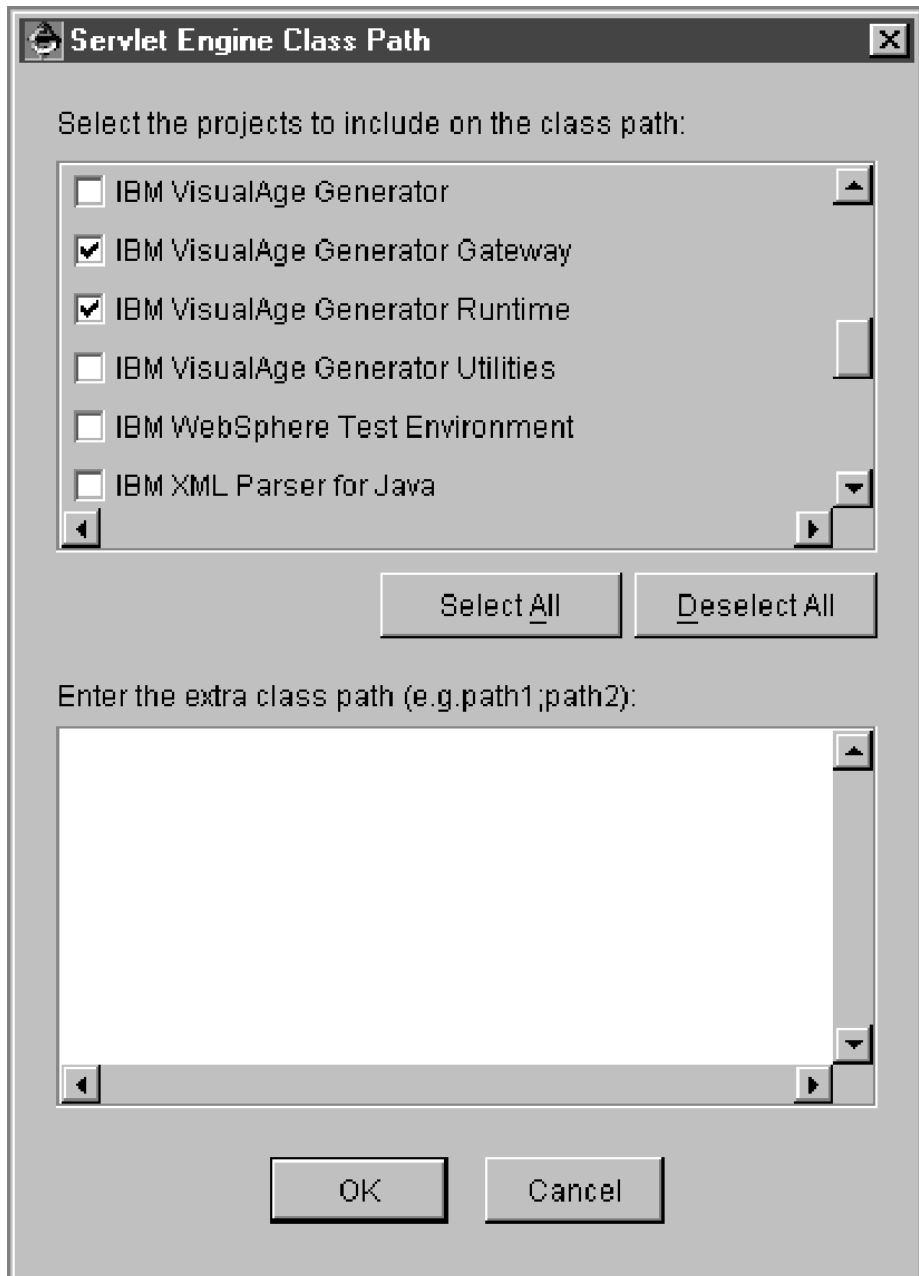


Figure 34. Servlet Engine Classpath dialog box

5. Select the checkboxes that refer to the runtime classes *IBM VisualAge Generator Gateway* and *IBM VisualAge Generator Runtime*.

Note: IBM VisualAge Generator Gateway is a project you created, as described in “WebSphere Test Environment” on page 54.

6. If you are using the CICS Transaction Gateway to interact with a remote CICS catcher, add the fully qualified path for files `ctgclient.jar` and `ctgserver.jar`, which are in the CICS installation `CLASSES` subdirectory.
7. Click **OK**.

Complete the general set up

To complete the general set up:

1. To configure the gateway in general, follow the directions in “Appendix A. Configuring the Gateway Servlet” on page 117. To configure the gateway servlet specifically for Callable ITF (as is recommended), follow the directions in “Configuring the gateway servlet” on page 52.
2. Copy the gateway-servlet JSPs from the installation directory (usually, `C:\lbmvagen\Vgwgs45`); the files are `CSOERRORUIR.jsp`, `Vagen1EntryPage.jsp`, `Vagen1ErrorPage.jsp`, and `Vagen1LogonPage.jsp`. Place those files in the document-root directory; in `c:\webroot1`, in this example.

Deploy web-transaction files to tier 2

This section gives information you can use when you are setting up either WTE or a production-level web application server.

Use the following naming conventions to identify the tier 2 objects generated for a web transaction:

- Edit table—
`VGDataedit_table_name.tab`
`VGTbledit_table_name.java`
- Message resource bundle—
`message_table_prefixRBundlelocale.java`

The *locale* may be null.

- UI record bean—
`UI_record_nameBean.java`
- UI record resource bundle—
`UI_record_nameRBundlelocale.java`

The *locale* may be null.

- UI record JSP—
`UI_record_name.jsp`
- UI record object—
`VGUIrUI_record_name.java`

The file extension for a compiled Java part is `.class`, and files with that extension are deployed on tier 2, along with the `jsp` and `tab` files. The class and `tab` files are deployed in the class directory (`c:\webclass1`, for example). The `jsp` files are deployed in the document-root directory (`c:\webroot1`, for example).

If you have specified generation option `PACKAGENAME`, the class files must be in a subdirectory whose name structure reflects the package name. If package name is *my.pkg*, for example, place the class files in directory `c:\webclass1\my\pkg`.

You can configure VisualAge Generator so that the prepared tier 2 objects are automatically placed in specific directories on the same or a different machine:

1. If your web application server is on a remote machine, specify the generation options `JVADESTHOST` and `JVADESTDIR`. For details, see the *VisualAge Generator Generation Guide*.
2. If you are preparing files by use of a command file that executes on the machine where the web application server resides:
 - a. Set environment variable `HPTCLASSDIR` to the fully qualified path of the top-level directory where class files should go. In our example, the directory value is `c:\webclass1`.
 - b. Set environment variable `HPTJSPDIR` to the document root for the web application. In our example, the directory value is `c:\webroot1`.

Running a web transaction in WTE

In WTE as in all tier 2 environments, do the following before running a generated web transaction:

1. Start the Session ID Manager.
2. Start the web application server.
3. Invoke the gateway servlet.

You also must start a catcher program—in most cases, a TCP/IP listener—on the third tier.

Start the Session ID Manager

The Session ID Manager is included in the jar file that contains the gateway servlet, but is run separately from the gateway servlet. As noted in “WebSphere Test Environment” on page 54, you installed that file into a VisualAge project called *IBM VisualAge Generator Gateway*. To start the Session ID Manager:

1. Go to the Workbench project list.
2. Expand IBM VisualAge Generator Gateway.
3. Select package `com.ibm.hpt.gateway`.
4. Select the `SessionIDManager` class.

5. Click the Run icon.

Start the web application server

To start WTE, do the following:

- From the Workbench, click **Workspace→Tools→WebSphere Test Environment**.
- The WebSphere Test Environment Control Center is displayed (Figure 33 on page 58).
- If the leftmost pane does not display **Servlet Engine**, click **Servers**.
- Click **Servlet Engine**, then **Start Servlet Engine**.
- Continue the procedure only when the **Stop Servlet Engine** button is enabled.

Invoke the gateway servlet

To invoke the gateway servlet:

1. Open a browser.
2. Type the URL and press ENTER. The URL for the example—
`http://localhost:8080/Practice/GatewayServlet`
3. The default entry page JSP presents the default VisualAge Generator system entry page.

Start a catcher program on tier 3

A catcher program must be running on each platform that has a web transaction that you want a user to invoke.

In most cases, the catcher is a TCP/IP listener. If tier 3 is Callable ITF, for example, start a TCP/IP listener by following the procedure in “Assigning a TCP/IP port to Callable ITF” on page 51. For information on other platforms, see “Configuring tier 3” on page 123.

Chapter 5. Java Server Pages and the UI Record Bean API

This chapter describes how to customize the default JSPs generated by VisualAge Generator.

For example enhancements of generated JSPs, see *Building Enterprise Web Transactions using VisualAge Generator JavaBeans and JSPs*, which is cited in the Preface. Also cited there is a useful introduction to JSPs: *Servlet and JSP Programming with IBM WebSphere Studio and VisualAge for Java*.

JSP syntax

JSPs are ASCII files that have a .jsp suffix and contain a mixture of regular HTML tags and JSP syntax. A JSP is taken by IBM WebSphere Application Server and transformed into a Java servlet by a process known as page compilation.

VisualAge Generator currently supports the JSDK 2.1 and JSP 1.0 specifications. The full details of these specifications and associated syntax can be downloaded from the Sun web site for Java:

<http://www.javasoft.com>

All IBM WebSphere Application Server version 3.x platforms (standard, advanced, and enterprise) support JSDK 2.1 and either JSP 0.91 or JSP 1.0. IBM WebSphere Application Server version 2.x supports only JSDK2.0 and JSP 0.91.

While VisualAge Generator may provide support for JSDK 2.0 in a fixpak, the JSPs generated by VisualAge Generator will always be at the JSP 1.0 level.

In the remainder of this chapter we will look at the most commonly used features of the JSP 1.0 syntax. The text indicates where syntax is invalid at the JSP 0.91 level.

Scriptlets

A scriptlet can be inserted in a JSP with the following syntax:

```
<% --- any legal Java code --- %>
```

Here Java code is enclosed in <% %> symbols. At runtime the Java executes dynamically on the server side, nothing of it is seen in the HTML stream sent to the browser once the JSP completes running, except the results of Java statements to print to the special "out" PrintWriter object.

The "out" object is created and made available to the scriptlet as part of the page compilation process of the JSP.

Scriptlets are typically used to code some logic to selectively decide what to print to "out". An example of a very simple scriptlet is shown below:

```
<% if (x == 1) {out.println(EMPNO.getLabel());} %>
```

Scriptlets may be placed anywhere in the JSP source. What they print will be inserted into the HTML stream at that point.

Valid print operations include `out.print(string_value)` and `out.println(string_value)`. `println` puts a carriage control character on the end, an addition that makes the HTML source look tidy when viewed in a browser. If the user asks to view source, the carriage return does not have the effect of HTML `<BR>`.

What you print should be an object of class `String`. You can convert all objects (other than primitives) which are not already `Strings` to `strings` via a `toString` method. It is up to the object creator to decide whether to provide their own implementation of the method or inherit a `toString` from a class above it in the inheritance hierarchy. (All classes you define in Java extend (or inherit) characteristics from some object, the `Object` class is the root.) If the object inherits a `toString` method, then what you see when running that method might not be exactly what you desire.

Primitives can also be converted to `Strings` by using their class wrappers, for example, an `int` can be converted to a `string` using a static method of the `Integer` class: `"String myConvertedString = Integer.toString(anInt);"`

We can use the UI record bean API documented in section "The UI record bean API" on page 67 to obtain access to the objects which represent data items in a UI record and to print them.

Scriptlet code all ends up inside the same Java method of the page compiled JSP (the "service" method). Thus objects created in one scriptlet can be referenced by another.

To write scriptlets other than very simple ones requires a basic knowledge of Java.

Expressions

An expression can be inserted in a JSP with the following syntax:

```
<%= --- an object reference --- %>
```

These expressions are a shorthand way of simply printing the object to the HTML output stream sent to the browser. The object is automatically converted to a string for you and then printed, all you need to code is an object reference.

The expression

```
<%= anObject %>
```

(without a semicolon) is equivalent to the expression

```
<% out.print(anObject.toString()); %>
```

Expressions can be inserted anywhere in the JSP source and their print output is interleaved with the HTML surrounding it. They end up inside the same Java method of the page compiled JSP as scriptlet (the "service" method).

Most dynamic content equates to the insertion of a few expressions among regular HTML and client side scripting.

If you have indexed properties (that is properties which occur more than once) you will need to use scriptlets to code a Java "for" or "while" loop, to cycle through the occurrences. Remember that Java indices start at 0.

To build loops without coding in Java, you can use an IBM WebSphere Application Server version 3.x-specific JSP tag:

```
<tsx:repeat  
  index=anIndexName start=startingIndex end=endingIndex>  
....  
</tsx:repeat>
```

The equivalent IBM WebSphere Application Server version 2.x tag:

```
<repeat index=ind start=startingIndex end=endingIndex>  
.....  
</repeat>.
```

Bean tag

A bean tag can be inserted in a JSP with the following syntax:

```
<jsp:useBean id="referenceName" class="fullyQualifiedClassnameOfThisBean" scope="?" />
```

This syntax is not valid with the 0.91 JSP specification, although there is an equivalent tag.

The bean tag allows the JSP to reference or even create a Java bean. Once we have the bean we can reference or change its properties.

The bean tag does not have to be empty, it can have a closing tag.

The `id` attribute defines the name you use to reference this bean elsewhere in the JSP inside scriptlets or expressions.

The `scope` attribute can equal any of the following:

- **session**, which means the UI record bean was stored in the `HttpSession` object.
- **request**, which means the UI record bean was stored in the `HttpServletRequest`. (The gateway servlet stores the UI record bean there.)
- **page**, —which means the bean was stored in the JSP page context
- **application**, which means the bean was stored in the servlet context

Refer to the Sun documentation for more details on `page` and `application`; they are included here for completeness.

There is also a **type** attribute, which represents an interface implemented by the bean and can be used instead of or in addition to the class.

In our case a reference to the UI record bean for our UI record for an individual browser user is passed to us by the gateway servlet by storing it in the `HttpServletRequest` object which the Java system provides to the servlet as part of the JSDK architecture. There is a separate `HttpServletRequest` object for each browser user and this object is passed to the JSP when it is invoked by the gateway servlet.

An example bean tag for the JSP generated from the UI record named `JANEUI1` is as follows:

```
<jsp:useBean id="JANEUI1" scope="request" class="my.pkg.JANEUI1Bean" />
```

This tag appears near the beginning of the JSP source.

We could now use `JANEUI1` to get a reference to UI record data items, but this referencing is done in the generated JSP, straight after the bean tag; for example:

```
<% VGDataElement TXTFLD = JANEUI1.getTXTFLD(); %>
```

All we have to do to refer to the data item `TXTFLD` is to use the name `TXTFLD`. There are several methods that are listed in section “`VGDataElement` Interface” on page 68. To display the label we defined in the text field’s UI properties, for example, we could code the following:

```
<%= TXTFLD.getLabel() %>
```

Directives

A directive can be inserted in a JSP with the following syntax:

```
<%@ page aValidDirective %>
```

This syntax describes JSP wide controls.

This syntax is not valid with the 0.91 JSP specification, although there are equivalents to some of the attributes.

Directives used in JSPs generated by VisualAge Generator include the following:

import

```
<%@ page import="partial package name" %>
```

With import you can add Java import statements which will apply to any scriptlets or expressions anywhere inside the JSP. Java import statements are a shorthand to save you from having to type out the fully qualified name of the package everywhere when referring to elements inside it.

VisualAge Generator generates the following into the JSPs for UI records:

```
<%@ page import = "com.ibm.vgj.uibean.VGDataElement" %>
```

errorPage

```
<%@ page errorPage="jspName" %>
```

This allows you to specify a JSP to show if this JSP throws an unhandled exception.

The UI record bean API

The String class is used by all get and set methods implemented for data items that are defined with a User Interface Type of **Input/Output** in the Java Bean generated for the UI record.

Set and get methods implemented for data items that are defined with a User Interface Type of **Nonereturn** the appropriate Java class for the item.

UI Record Bean Interface

The get and set methods implemented for the UI record bean are described in this section.

String getTitle();

Returns title property of UI record.

String getHelpText();

Returns help text property of UI record.

String getGatewayURL();

Returns the gateway URL and is used as the ACTION of an HTML form.

String getSecureGatewayURL();

Returns the gateway URL but with HTTPS protocol and is used as the ACTION of an HTML form.

String getPageID();

String form of number that uniquely marks the page that is served to the client.

String getAppID();

Returns the ID that identifies the web transaction with which the UI record is associated.

String getSessionID();

Returns the ID that identifies the current gateway session that will process the submit request.

boolean hasInputError();

Indicates whether or not any item in the record is in error.

VGDataElement elementNamed(String name);

Returns element in UI record bean named *name*.

VGDataElement get<item name>();

Generated get methods for each UI record bean instance.

void set<item name>(String value);

Generated set methods for each UI record bean instance.

VGDataElement Interface

Methods available in the VGDataElement interface are described in this section.

Enumeration getEditTableValues();

Returns the elements in an edit table associated with an input field.

String getErrorMessage();

Returns the error message associated with the element.

String getGatewayURL();

For items that are not of UI type Form or Program Link, this will return the same value as the UI record bean version of this method.

However for items that are of UI type Form or Program Link this method returns a URL string that contains all the parameters as defined by the link properties. This string is then usable as an HREF in an <A>HTML element.

String getSecureGatewayURL();

Same as getGatewayURL() above but with the HTTPS protocol.

String getHelpText();

Returns the help text property of the element.

int getIndex();

Returns the current occurrence of an element which occurs.

String getLabel();

Returns the label UI property of an item. If item is an element of an array, the label defined for the index of the VGDataElement instance will be returned.

String getTextValue();

Returns String value of element with all output formatting on data done.

TableModel getTextValuesTable();

Returns a TableModel of all formatted text values for the occurrences and sub-elements of the VGDataElement instance.

boolean hasInputError();

Returns whether or not the element has an input error.

boolean isDisplayable();

Returns *true* if the data item associated with a submit button has a non-blank value.

boolean isEmpty();

Returns *false* if the item is not occurred or if the item is occurred but has no Occurrences item. Returns *true* if zero is the value of the related Occurrences item.

For details on the Occurrences item, see “Control Items” on page 27.

boolean isSelected();

Returns *true* if the index of the element is a value in the associated **Selected Index Item**.

Enumeration occurrences();

Returns Enumeration of VGDataElements for each valid index (index <= numOccurs item value). If occurs value is 1, this will be a single element Enumeration.

Enumeration subElements();

Returns Enumeration of VGDataElement that are valid sub-elements (not of UI type None) of the VGDataElement instance. Only the lowest level sub-elements will be returned. The index of each sub-element is that of the VGDataElement instance.

void setDateTimeFormat(*DateFormat_object*);

Sets a Java DateFormat object to specify the valid format for datetime values that pass between the browser and tier 2 in either direction. You can use this method only on a data item that has been assigned a date or time edit.

Chapter 6. Reviewing Sample Web Transactions

A number of web transactions are installed with VisualAge Generator, along with database-creation files and a set of database-access programs of type Called Batch. By reviewing that material, this chapter helps you to understand alternative ways to structure a web transaction and to manage program state. Subjects include the following:

- Defining and testing web transactions.
- Customizing the UI record for different output.
- Using the input-edit capabilities of the UI record data items.

Set up

This section describes how to set up your environment.

Extracting the sample files

The files necessary to run the sample programs are found in the self-extracting file VGV45WEB.EXE. The files include the following:

- VisualAge Generator code
- DB2 database materials
- WebSphere Studio project archives
- Additional runtime files

To extract the sample files, do the following:

1. Run VGV45WEB.EXE.

You will be prompted for the destination directory.

2. Select the base directory where you want to store the sample files and select **OK**.

Several directories will appear in the base directory that you selected along with `readme.txt`. Open `readme.txt` to view information about the sample files you extracted.

Setting up the database

The sample programs reference DB2 tables in the ITSOBANK database. To create the necessary tables, do the following:

1. From a command prompt, type `db2cmd`

A new prompt window titled **DB2 CLP** will appear.

2. In the new prompt window, change the current directory so that you are in the directory containing the database files.

The files will be located in
<localdrive>:\<base_path>\Database

where <localdrive> and <base_path> are the drive and directory path to which you extracted VGV45WEB.EXE.

3. At the command prompt, type BankDB.bat.

The command file will first attempt to remove the ITSOBANK database. It will then create the ITSOBANK database and a series of SQL commands will be run to initialize the database.

4. At the command prompt, type WTSDB.bat.

The WT_STATE table is created. You have completed the database setup.

Loading the code

To review the programs, you need to load the sample code into your repository and add it to your workspace.

To load the code, do the following:

1. From the Workbench, choose **Selected→Import...**
2. Select the radio button labeled **Repository** as the import source and click on the **Next >** push button.
3. Choose **Local repository** as the import option and select **Browse** to locate the following file:

<localdrive>:\<base_path>\VAGen\WebTran.dat

WebTran.dat contains the sample programs.

4. Select the **Project** radio button and select the **cDetails...** push button next to it.

The Project import window appears.

5. In the Projects pane select the checkbox next to
VGV45 Redbook WebTran Programs

In the Versions available pane, *1.0 is automatically checked.

6. Select **OK**.

The Project import window is closed.

7. Select the **Add most recent project edition to workspace** checkbox and select **OK**.

Selecting the checkbox means that the project will be automatically added to the workspace.

The Workbench Projects pane will now display a project called
VGV45 Redbook WebTran Programs 1.0

If you expand the project listing, a package called

will be listed as well.

Program structure

This section illustrates the basic approaches to developing web transactions.

CONVERSE

The first approach you will examine is a web transaction program (CSTCNV) that uses a CONVERSE to prompt the user with a Customer Info Web page implemented using a UI record (CUSTUI).

The structure of this system is shown in Figure 35.

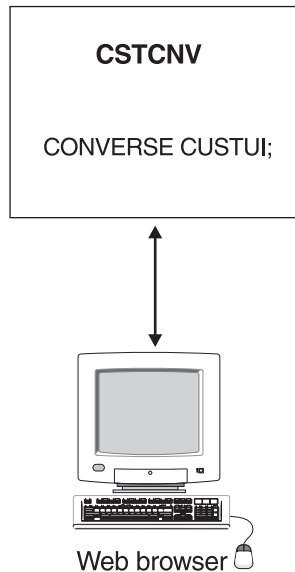


Figure 35. CSTCNV Web Transaction program structure

This Customer Info Web page allows the user to enter a customer ID and click a button to invoke a server that searches the database to find the customer record. The record data is returned if found, otherwise an error message is displayed.

UI record input edits are used to verify that input error checking takes place before invocation of the server.

A UI record (CUSTUI) that will be used by the CONVERSE model web transaction program CSTCNV has been defined. The UI record definition implements the user interface shown in Figure 36.

Customer Info - Netscape

File Edit View Go Communicator Help

Bookmarks Location: ay?hptAppId=CSTCNV What's Related

Customer Info

Customer ID:

First Name:

Last Name:

Bank ID:

ENTER CUSTOMER ID AND PUSH FIND.

Document: Done

Figure 36. Customer Info Web page

Customer info UI record

CUSTUI is the UI record that defines the basic properties of the user interface that is implemented in a Web browser. The CUSTUI record is shown in Figure 37 on page 75.

CUSTUI - Record Editor
File Edit View Define Tools Help

User Interface Default Usage: Nonshared

Name	Occurs	Type	Length	Decimals	UI Type	UI Properties	Usage	Description
BUTTON1	1	Char	8	0	Submit	Custom	Nonshared	
BUTTON2	1	Char	8	0	Submit Bypass	Custom	Nonshared	
BUTTON-VALUE	1	Char	8	0	None	None	Nonshared	
CUSTID	1	Char	4	0	Input/Output	Custom	Nonshared	
FNAME	1	Char	30	0	Input/Output	Custom	Nonshared	
LNAME	1	Char	30	0	Input/Output	Custom	Nonshared	
BANKID	1	Char	4	0	Input/Output	Custom	Nonshared	
MESSAGE	1	Char	70	0	Output	Default	Nonshared	

Figure 37. CUSTUI UI record definition

The record is defined as a User Interface in the record type definition drop-down list. The data items in the record are defined as well. The data items include the fields and buttons which are displayed in the web page.

Customer info web transaction

The CSTCNV web transaction program implements the CONVERSE model. There is a main function associated with the program called CSTCNV-MAIN. This function has an I/O option (CONVERSE) and an I/O object (CUSTUI) defined. The entire function logic is shown in Figure 38.

CSTCNV-MAIN - Function Editor
File Edit View Define Tools Help

CONVERSE CUSTUI 001:001

```

MESSAGE = 'ENTER CUSTOMER ID AND PUSH FIND.';
WHILE BUTTON-VALUE ^= 'EXIT';
  ***** CONVERSE CUSTUI *****
  IF BUTTON-VALUE = 'FIND';
    MESSAGE = '';
    CALL CUSTPGM CUSTUI;
  END;
END;

```

Figure 38. CSTCNV-MAIN processing logic

Notice that the line CONVERSE CUSTUI appears in the code as a result of the selection of I/O option and I/O object.

Invoking and testing the customer info program

You can see the CONVERSE model in action by testing the sample program.

1. Click on the Test icon in the CSTCNV program editor.

The Test Monitor window appears.

2. Use the test facility step action to walk through the specified processing logic (see Figure 38 on page 75).

When the CONVERSE option is encountered, the test facility will invoke the Web browser and dynamically build the HTML required for the CUSTUI UI record definition (at runtime this is done by a generated JSP).

In the browser we see the Customer Info Web page with labels (UI type of output) and input fields (UI type of input or input/output).

3. View the HTML source for the browser display. If options such as **View→Page** source do not work, try saving the current page source to a file and viewing it with a text editor.

You should be able to see a form with text and input field components implemented from the CUSTUI UI record definition.

4. Enter a valid **Customer Id**, for example, the value 101, and select **Find**.

You should see how the program loops back on the CONVERSE with the information found in the database.

- Enter additional **Customer Id** values or select the **Exit** button to end the web transaction program.
- Use the test facility to check the UI record data item input values when interacting with the web transaction.

Note: You can restart the test facility by clicking on the Test icon when the CSTCNV program is selected in the VAGen Parts window.

XFER with a named program

A UI record can be sent to a browser using a CONVERSE or an XFER statement. In this section, single segment programs, which use an XFER approach, are reviewed.

Using one UI record

There are multiple web transaction structure options available. The next option we will review results from converting the CONVERSE model CSTCNV program (as shown in Figure 35 on page 73) to a program implemented using a single segment (XFER PGM) structure (see Figure 39 on page 77).

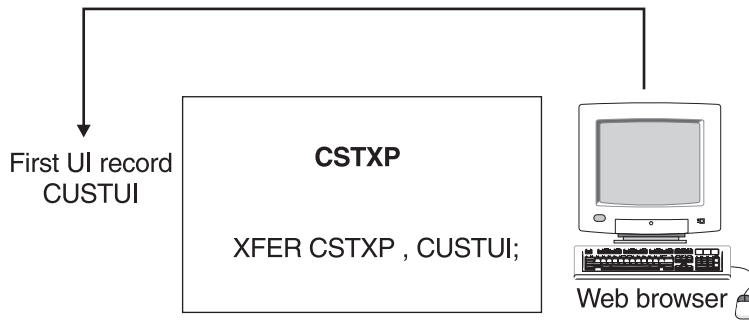


Figure 39. CSTXP Web Transaction program structure

The XFER PGM logic shown in Figure 39, also known as "XFER with a named program," does not include the use of a working storage record (the full syntax is XFER PGM WSRec, UIRec). When a working storage record is not passed, only the data sent out in the UI record can be returned to the target web transaction program; which data is returned depends on the data item properties in the UI record.

Customer info UI record: The CUSTUI record in this model differs in two ways from the CUSTUI record in the CONVERSE model.

- Input to the Customer Id field is required before the XFER is run
By defining a minimum input value for the CUSTID data item, input is required by the program.
- The Exit button works without any check for input to the Customer Id field
By changing the UI Type of the Exit button to *Submit Bypass*, no check is made on the input.

The resulting record definition is shown in Figure 40.

CUSTUI - Record Editor								
File Edit View Define Tools Help								
					User Interface	Default Usage: Nonshared		
Name	Occurs	Type	Length	Decimals	UI Type	UI Properties	Usage	Description
BUTTON1	1	Char	8	0	Submit	Custom	Nonshared	
BUTTON2	1	Char	8	0	Submit Bypass	Custom	Nonshared	
BUTTON-VALUE	1	Char	8	0	None	None	Nonshared	
CUSTID	1	Char	4	0	Input/Output	Custom	Nonshared	
FNAME	1	Char	30	0	Input/Output	Custom	Nonshared	
LNAME	1	Char	30	0	Input/Output	Custom	Nonshared	
BANKID	1	Char	4	0	Input/Output	Custom	Nonshared	
MESSAGE	1	Char	70	0	Output	Default	Nonshared	

Figure 40. CUSTUI UI record definition

Basic single segment program structure: A Web Transaction program called CSTXP implements the same functionality as the CSTCNV program using the XFER with a named program model.

- CUSTUI is defined as the first UI record for the program.
- CSTXP-MAIN is the associated main function.

The logic for this function reuses some of the logic that appears in Figure 39 on page 77 to implement the same processing function as provided by the CONVERSE program structure.

Note: The first time the main function is invoked, no button has been pushed, so the program must loop to itself.

The required function logic is shown in Figure 41.

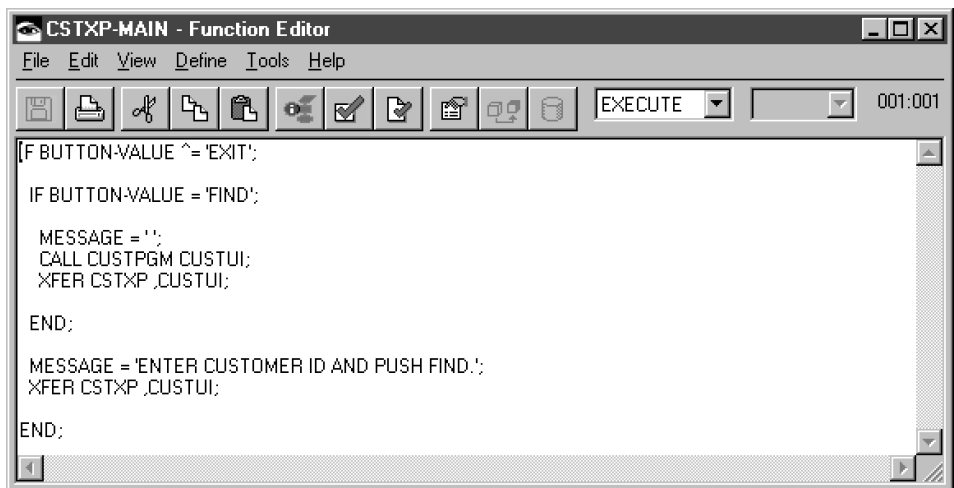


Figure 41. CSTXP-MAIN processing logic

Invoking and testing the customer info program:

1. Test the input edits and check the UI record input fields:
 - Enter a CustID value with only 2 digits and select **Find**. You should receive an error message in **red** that is positioned under the field in error. This is the default error management behavior associated with predefined edits. Additional customization is supported.
 - Enter invalid input and attempt to use the **Exit** button. Unlike with the **Find** button, the **Exit** button bypass definition allows the button to trigger the target program even when invalid input has been entered. A bypass edit button will not pass data values.

Using two UI records

A single segment web transaction program can be converted into a set of programs that use two UI records, one for end user input and one for end user output (see Figure 42).

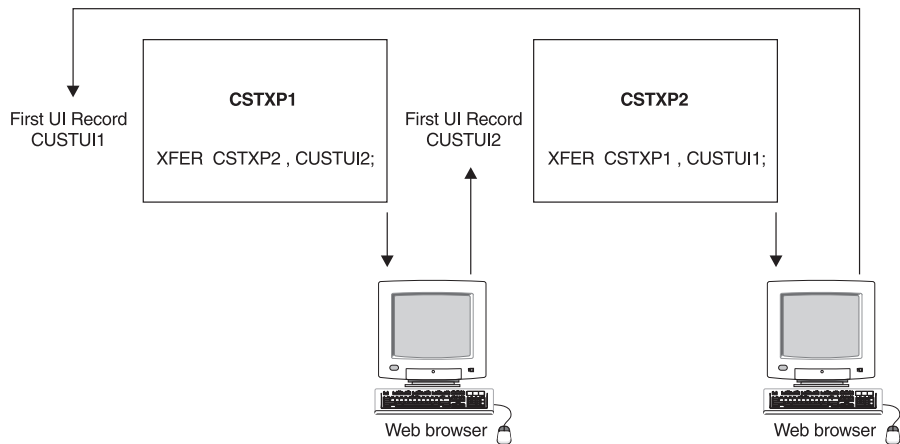


Figure 42. CSTXP1 and CSTXP2 Web Transaction programs structure

The UI record definition for end user input, CUSTUI1, implements the user interface shown in Figure 43 on page 80.

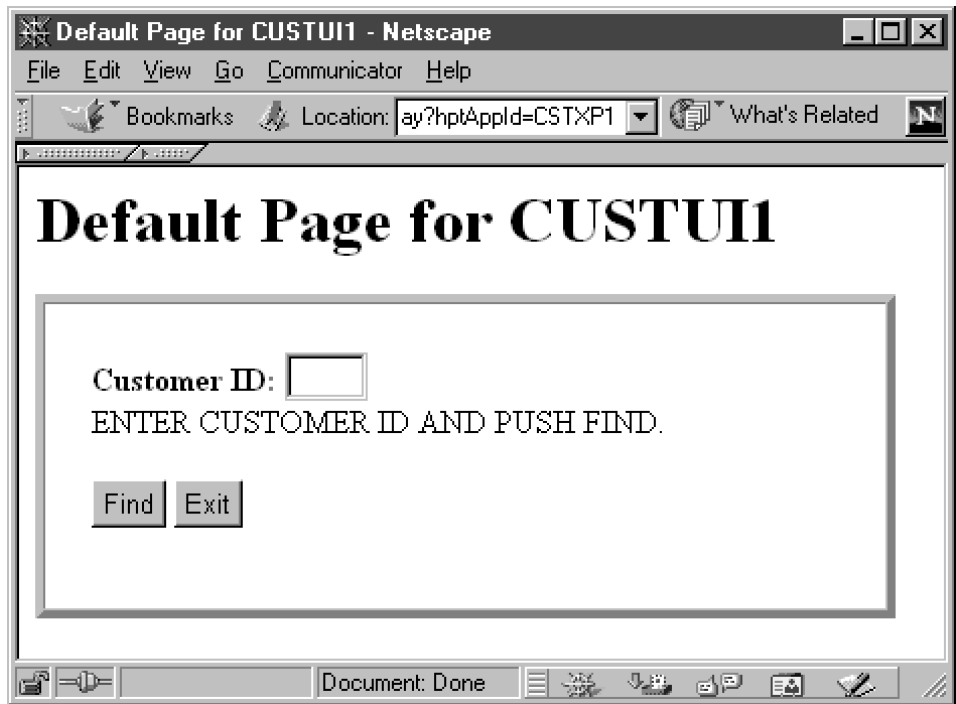


Figure 43. Input Customer Info Web page

The UI record definition for end user output, CUSTUI2, implements the user interface shown in Figure 44 on page 81.

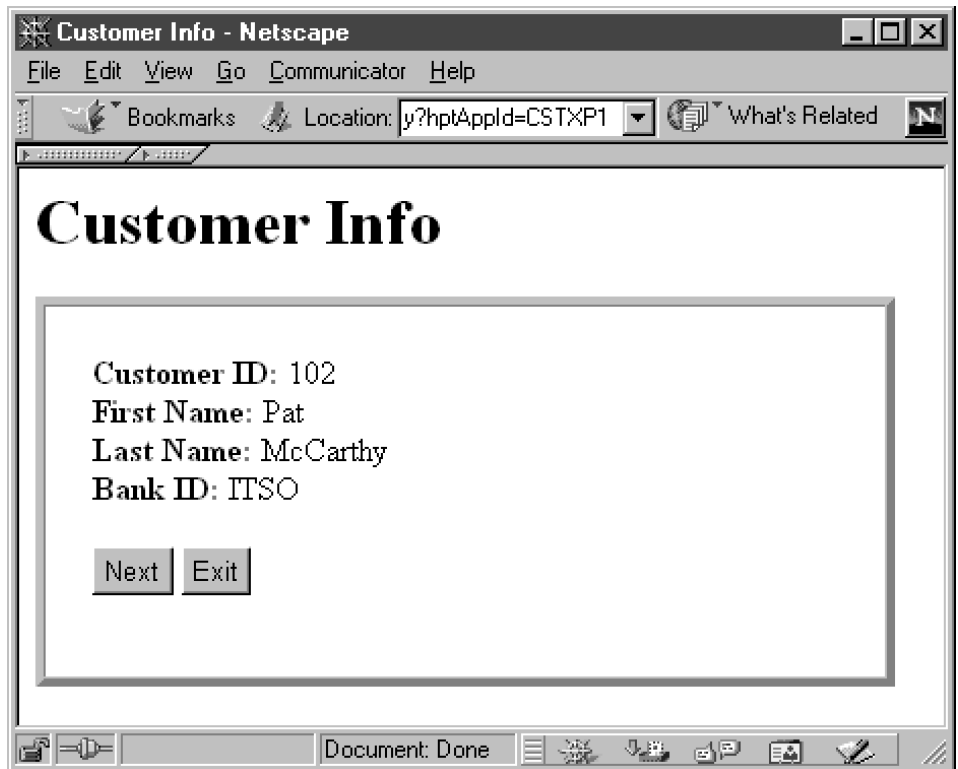


Figure 44. Output Customer Info Web page

The input and output customer info UI records: The UI records CUSTUI1 and CUSTUI2 are edited copies of the CUSTUI UI record. In record CUSTUI1:

- The CUSTID UI Type changed to Input
- Unnecessary data items removed

The resulting record definition is shown in Figure 45.

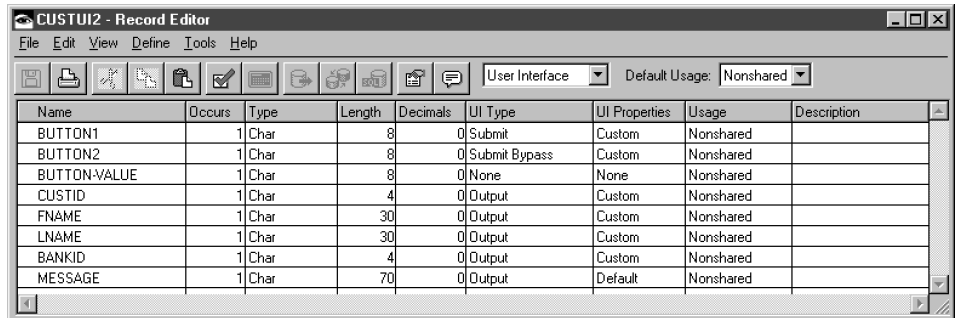
Name	Occurs	Type	Length	Decimals	UI Type	UI Properties	Usage	Description
BUTTON1	1	Char	8	0	Submit	Custom	Nonshared	
BUTTON2	1	Char	8	0	Submit Bypass	Custom	Nonshared	
BUTTON-VALUE	1	Char	8	0	None	None	Nonshared	
CUSTID	1	Char	4	0	Input	Custom	Nonshared	
MESSAGE	1	Char	70	0	Output	Default	Nonshared	

Figure 45. CUSTUI1 UI record definition

In record CUSTUI2:

- The UI Type for all database-only items (CUSTID, FNAME, LNAME, BANKID) changed to Output
- Minimum input value for CUSTID set to 0

The resulting record definition is shown in Figure 46.



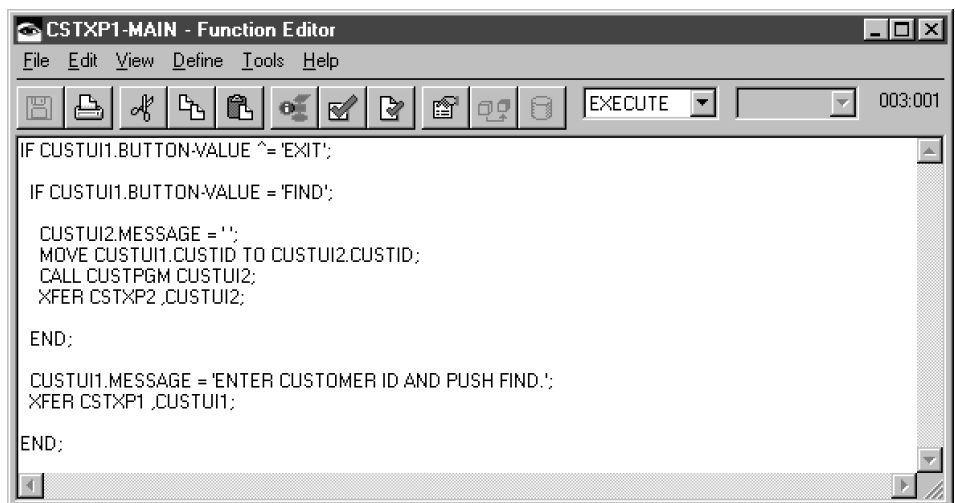
Name	Occurs	Type	Length	Decimals	UI Type	UI Properties	Usage	Description
BUTTON1	1	Char	8	0	Submit	Custom	Nonshared	
BUTTON2	1	Char	8	0	Submit Bypass	Custom	Nonshared	
BUTTON-VALUE	1	Char	8	0	None	None	Nonshared	
CUSTID	1	Char	4	0	Output	Custom	Nonshared	
FNAME	1	Char	30	0	Output	Custom	Nonshared	
LNAME	1	Char	30	0	Output	Custom	Nonshared	
BANKID	1	Char	4	0	Output	Custom	Nonshared	
MESSAGE	1	Char	70	0	Output	Default	Nonshared	

Figure 46. CUSTUI2 UI record definition

Single segment program structure with multiple UI records: CSTXP1 is the Web Transaction program that implements this model:

- CUSTUI1 defined as the first UI record for the program
- CSTXP1-MAIN is the main function.

The logic for CSTXP1-MAIN is shown in Figure 47.



```

IF CUSTUI1.BUTTON-VALUE ^= 'EXIT';
  IF CUSTUI1.BUTTON-VALUE = 'FIND';
    CUSTUI2.MESSAGE = '';
    MOVE CUSTUI1.CUSTID TO CUSTUI2.CUSTID;
    CALL CUSTPGM CUSTUI2;
    XFER CSTXP2 ,CUSTUI2;
  END;
  CUSTUI1.MESSAGE = 'ENTER CUSTOMER ID AND PUSH FIND.';
  XFER CSTXP1 ,CUSTUI1;
END;

```

Figure 47. CSTXP1-MAIN processing logic

Note: Short names for the data items do not work any more because there are two UI records associated to the web transaction. This means that function logic must fully qualify the data item name (i.e. CUSTUI1.BUTTON-VALUE).

The logic for CSTXP2-MAIN is shown in Figure 48.

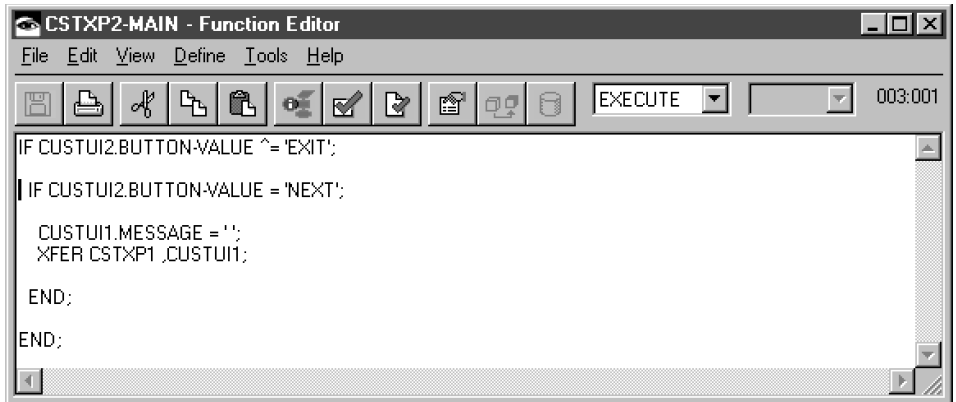


Figure 48. CSTXP2-MAIN processing logic

Use the test facility to test the CSTXP1 program (which transfers to the CSTXP2 program). Check the results for different input values when interacting with the web transactions.

Note: When using the XFER PGM statement and two different UI records (input and output views), two different web transaction programs are used. Only the first UI record defined for the target program can be referenced on the XFER statement, so one program is required for each UI record.

XFER with a blank

This section concerns a programming model known as "XFER with a blank." It is implemented with UI records that have defined forms. This programming structure can be used with one or more than one UI record (different input and output views) per web transaction.

Using one UI record

The same processing provided in the CONVERSE model CSTCNV program in a web transaction program can be implemented using a single segment structure (see Figure 49 on page 84).

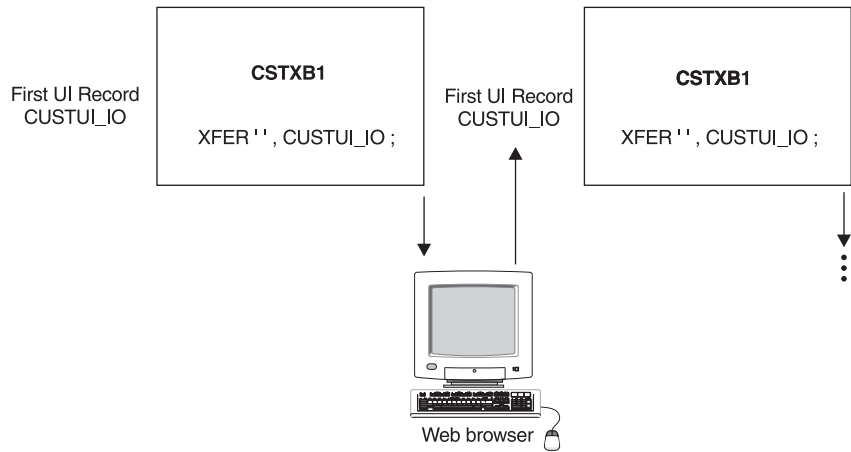


Figure 49. CSTXB1 Web Transaction program structure

The UI record definition for input and output, `CUSTUI_IO`, implements the user interface shown in Figure 50 on page 85.

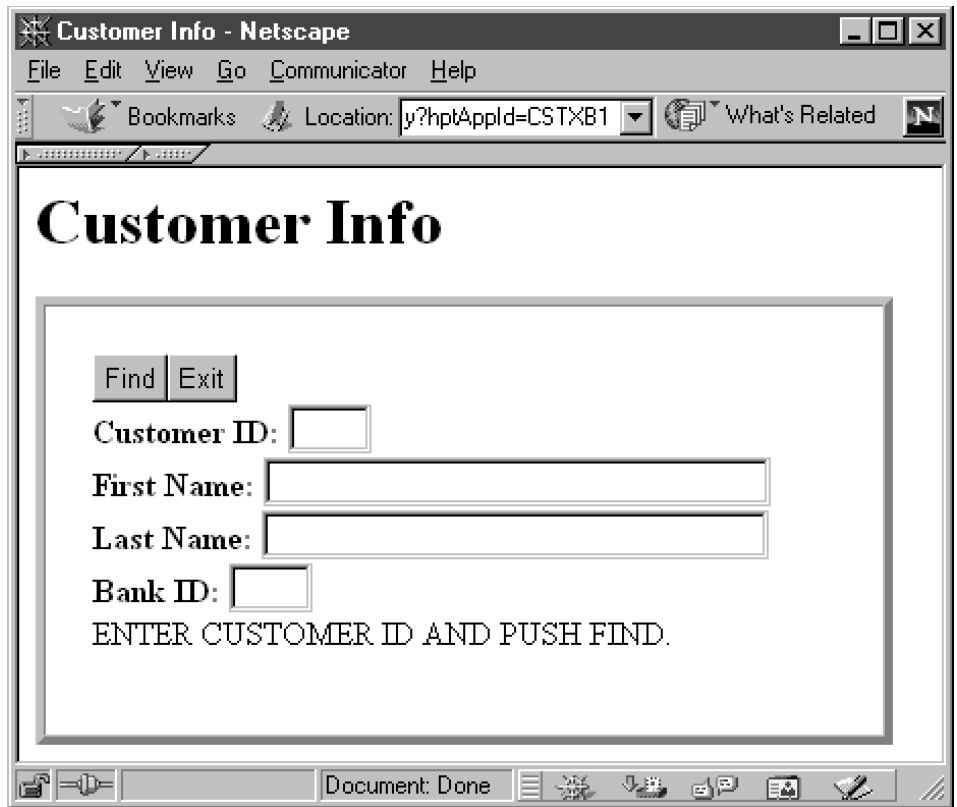


Figure 50. Customer Info Web page

Input and output customer info UI record: The UI record CUSTUI_IO is a copy of the CUSTUI2 UI record with several changes:

- The UI types of CUSTID, FNAME, LNAME and BANKID are Input/Output type.
- The CUSTID data item has a 3-character minimum input.
- CUSTUI_IO has a new data item of UI Type Form called FORM_IO. This form is associated with program CSTXB1 through a Program Link. All data items are substructures of FORM_IO.

The resulting record definition is shown in Figure 51 on page 86.

Name	Occurs	Type	Length	Decimals	UI Type	UI Properties	Usage	Description
FORM_IO	1	Char	162	0	Form	Custom	Nonshared	
BUTTONS	1	Char	16	0	Output	Default	Nonshared	
BUTTON1	1	Char	8	0	Submit	Custom	Nonshared	
BUTTON2	1	Char	8	0	Submit Bypass	Custom	Nonshared	
BUTTON-VALUE	1	Char	8	0	None	None	Nonshared	
CUSTID	1	Char	4	0	Input/Output	Custom	Nonshared	
FNAME	1	Char	30	0	Input/Output	Custom	Nonshared	
LNAME	1	Char	30	0	Input/Output	Custom	Nonshared	
BANKID	1	Char	4	0	Input/Output	Custom	Nonshared	
MESSAGE	1	Char	70	0	Output	Default	Nonshared	

Figure 51. CUSTUI_IO UI record definition

Single segment program structure: CSTXB1 is the web transaction program that implements the single segment program structure using the "XFER with a blank" model:

- CUSTUI_IO defined as the first UI record
- CSTXB1-MAIN is the main function.

The required function logic is shown in Figure 52.

```

IF BUTTON-VALUE ^= 'EXIT';
IF BUTTON-VALUE = 'FIND';
    MESSAGE = '';
    CALL CUSTPGM CUSTUI_IO;
    XFER '',CUSTUI_IO;

END;

MESSAGE = 'ENTER CUSTOMER ID AND PUSH FIND.';
XFER '',CUSTUI_IO;

END;

```

Figure 52. CSTXB1-MAIN processing logic

Use the test facility to check the validity of different input values.

Using two UI records

Another approach involves one Web application that uses three different UI records: the first to collect user input (CUSTUI_I), the second shows the use of

the output (CUSTUI_O), and the third is defined as the first UI record for the web transaction program (see Figure 53).

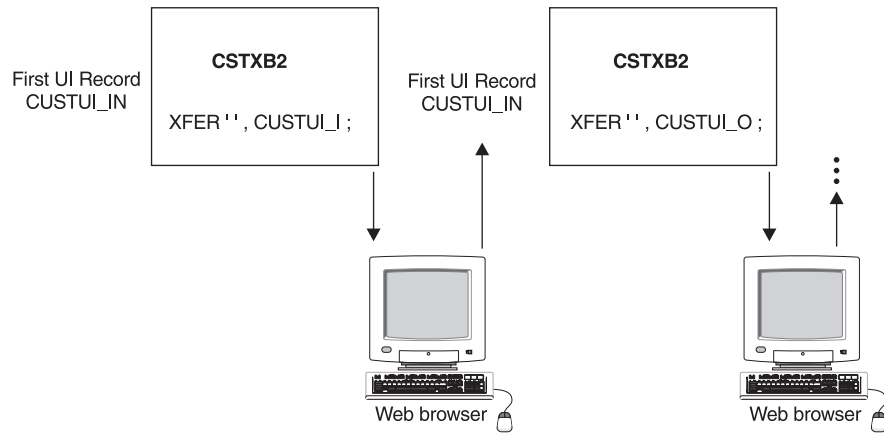


Figure 53. CSTXB2 Web Transaction program structure

The UI record definition for input, CUSTUI_I, implements the user interface shown in Figure 54.

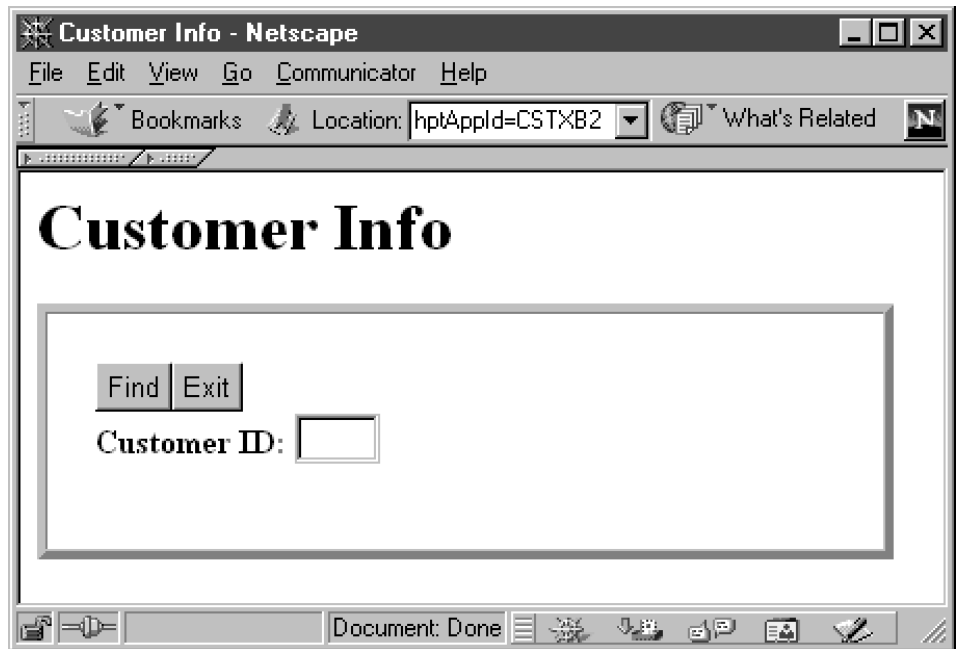


Figure 54. Input Customer Info Web page

The UI record definition for output, CUSTUI_O, implements the user interface shown in Figure 55.

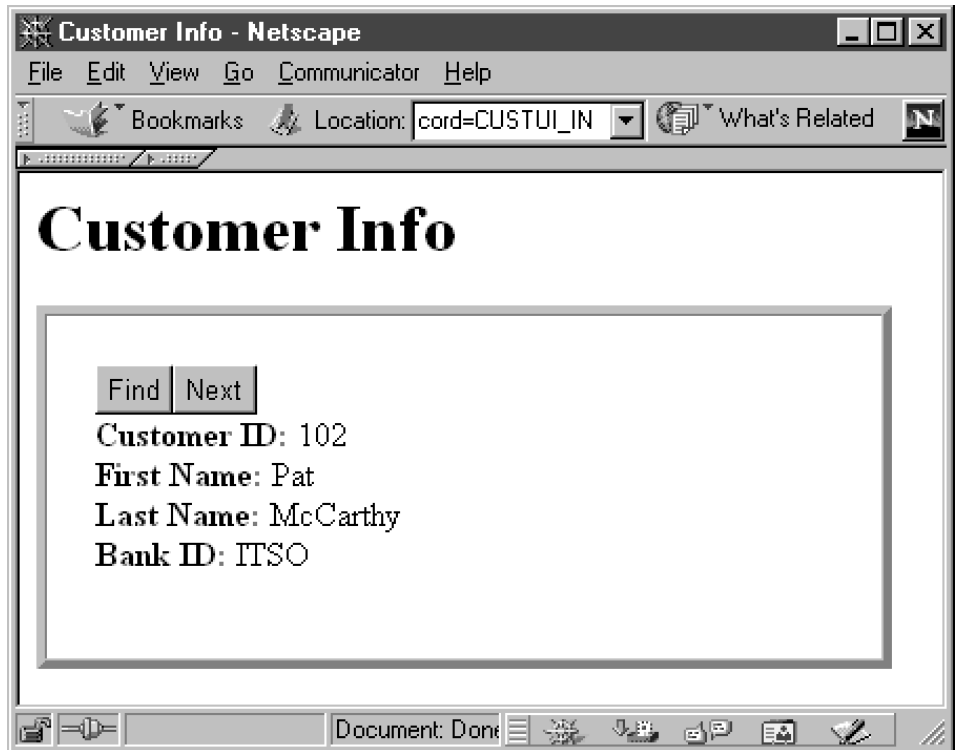


Figure 55. Output Customer Info Web page

Customer info UI records: The UI records CUSTUI_IN, CUSTUI_I, and CUSTUI_O are copies of the CUSTUI_IO UI record:

- Program CSTBX2 defined as Program Link by CUSTUI_I, CUSTUI_O, and CUSTUI_IN UI records
- CUSTUI_IN defined as the first UI record for CUSTUI_I, CUSTUI_O, and CUSTUI_IN UI records

The structure of the CUSTUI_I UI record is shown in Figure 56 on page 89.

The screenshot shows the 'CUSTUI_I - Record Editor' window. It has a menu bar (File, Edit, View, Define, Tools, Help) and a toolbar with icons for file operations and editing. Below the toolbar, there are dropdown menus for 'User Interface' and 'Default Usage: Nonshared'. The main area contains a table with the following data:

Name	Occurs	Type	Length	Decimals	UI Type	UI Properties	Usage	Description
FORM_I	1	Char	98	0	Form	Custom	Nonshared	
BUTTONS	1	Char	16	0	Output	Default	Nonshared	
BUTTON1	1	Char	8	0	Submit	Custom	Nonshared	
BUTTON2	1	Char	8	0	Submit Bypass	Custom	Nonshared	
BUTTON-VALUE	1	Char	8	0	None	None	Nonshared	
CUSTID	1	Char	4	0	Input/Output	Custom	Nonshared	
MESSAGE	1	Char	70	0	Output	Default	Nonshared	

Figure 56. CUSTUI_I UI record definition

The structure of the CUSTUI_O record is shown in in Figure 57.

The screenshot shows the 'CUSTUI_O - Record Editor' window. It has a menu bar (File, Edit, View, Define, Tools, Help) and a toolbar with icons for file operations and editing. Below the toolbar, there are dropdown menus for 'User Interface' and 'Default Usage: Nonshared'. The main area contains a table with the following data:

Name	Occurs	Type	Length	Decimals	UI Type	UI Properties	Usage	Description
FORM_O	1	Char	162	0	Form	Custom	Nonshared	
BUTTONS	1	Char	16	0	Output	Default	Nonshared	
BUTTON1	1	Char	8	0	Submit	Custom	Nonshared	
BUTTON2	1	Char	8	0	Submit Bypass	Custom	Nonshared	
BUTTON-VALUE	1	Char	8	0	None	None	Nonshared	
CUSTID	1	Char	4	0	Output	Custom	Nonshared	
FNAME	1	Char	30	0	Output	Custom	Nonshared	
LNAME	1	Char	30	0	Output	Custom	Nonshared	
BANKID	1	Char	4	0	Output	Custom	Nonshared	
MESSAGE	1	Char	70	0	Output	Default	Nonshared	

Figure 57. CUSTUI_O UI record definition

The structure of the CUSTUI_IN record is shown in Figure 58.

The screenshot shows the 'CUSTUI_IN - Record Editor' window. It has a menu bar (File, Edit, View, Define, Tools, Help) and a toolbar with icons for file operations and editing. Below the toolbar, there are dropdown menus for 'User Interface' and 'Default Usage: Nonshared'. The main area contains a table with the following data:

Name	Occurs	Type	Length	Decimals	UI Type	UI Properties	Usage	Description
FORM_IN	1	Char	106	0	Form	Custom	Nonshared	
BUTTONS	1	Char	24	0	Output	Default	Nonshared	
BUTTON1	1	Char	8	0	Submit	Custom	Nonshared	
BUTTON2	1	Char	8	0	Submit	Custom	Nonshared	
BUTTON3	1	Char	8	0	Submit Bypass	Custom	Nonshared	
BUTTON-VALUE	1	Char	8	0	None	None	Nonshared	
CUSTID	1	Char	4	0	Input/Output	Custom	Nonshared	
MESSAGE	1	Char	70	0	Output	Default	Nonshared	

Figure 58. CUSTUI_IN UI record definition

Single segment program structure with multiple UI records: CSTXB2 is the web transaction program that implements the single segment program structure with multiple UI records:

- CUSTUI_IN defined as the first UI record
- CSTXB2-MAIN is the main program.

The required function logic is shown in Figure 59.

Use the test facility to test the CSTXB2 program and check data values. Try

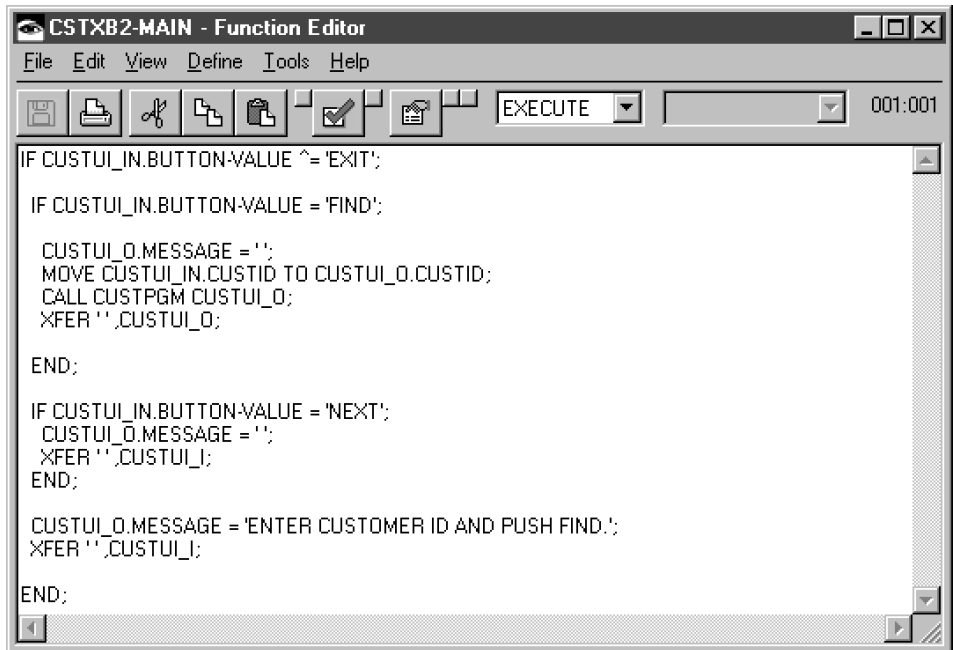


Figure 59. CSTXB2-MAIN processing logic

entering less than 3 characters in the CUSTID field. Notice that the error is returned in the CUSTUI_IN record, as all edits are processed in the first UI record defined for a single segment program.

Implementing global state management

It is common in application systems to have logic that implements different functions based on the type of user. In our database example, we can add control logic that allows update access for some users, so they can use the application to change the actual value of some fields in the database.

To keep the implementation simple, a predefined application will ask for the user name and then pass data back to the target web transaction for use in

determining application level authority. This passed information is the *state data* that must be kept active by all subsequent web transaction programs.

In traditional systems, this state data would be shared by all active programs, using techniques such as a passed common working storage record. There are multiple options available for storing state data in a web transaction system:

- UI record data kept either in session data or in hidden fields that are in the HTML sent to the browser and defined to return to the target program
- Working storage record(s) whose state is saved by VisualAge Generator runtime processing during user interaction
- Self-managed state processing

The implementation of this global data access in web transactions is studied in this section.

UI record-based state management implementation

State data can be kept active in the UI record used by the web transaction. The approach depends on the web transaction program structure.

CONVERSE

In this model, the CSTCNV web transaction is converted into one that accepts state information from the CSTIDUS web transaction. The structure of this system is shown in Figure 60 on page 92.

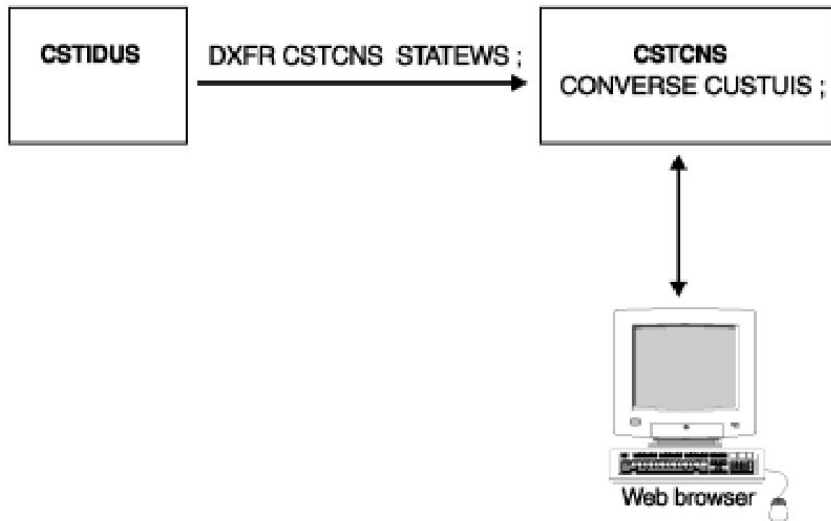


Figure 60. CSTCNS Web Transaction program structure

The UI record provided for input, CUSTUI_ID, implements the user interface where the user type and next web transaction are selected (see Figure 61 on page 93).

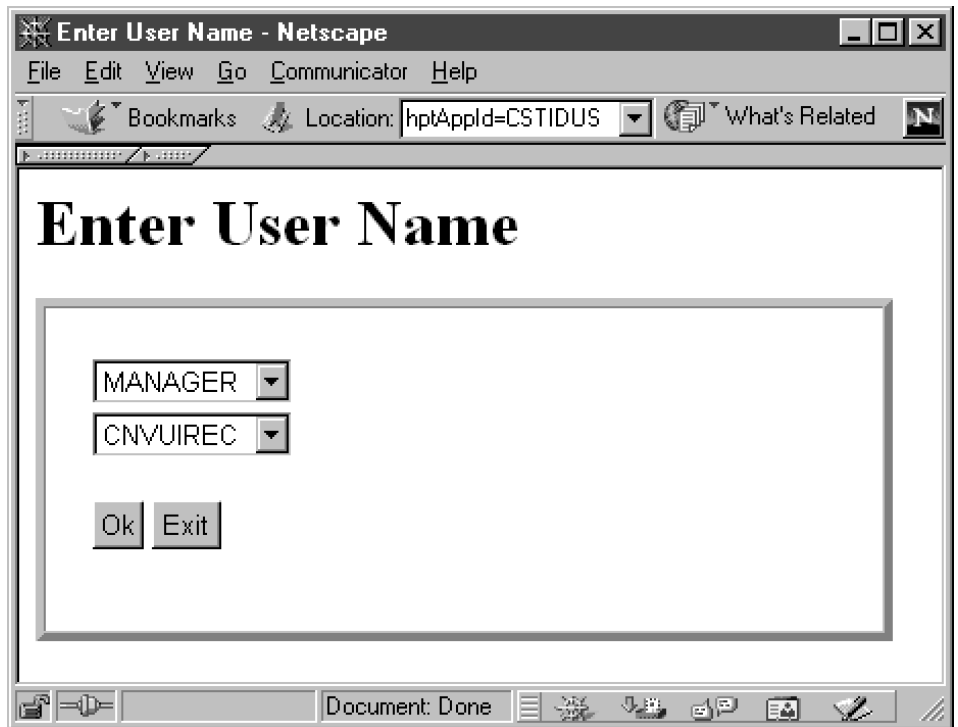


Figure 61. User Name inquiry Web page

The UI record definition for output, CUSTUIS, implements the user interface shown in Figure 62 on page 94, without update support, or the user interface shown in Figure 63 on page 95, with update support.

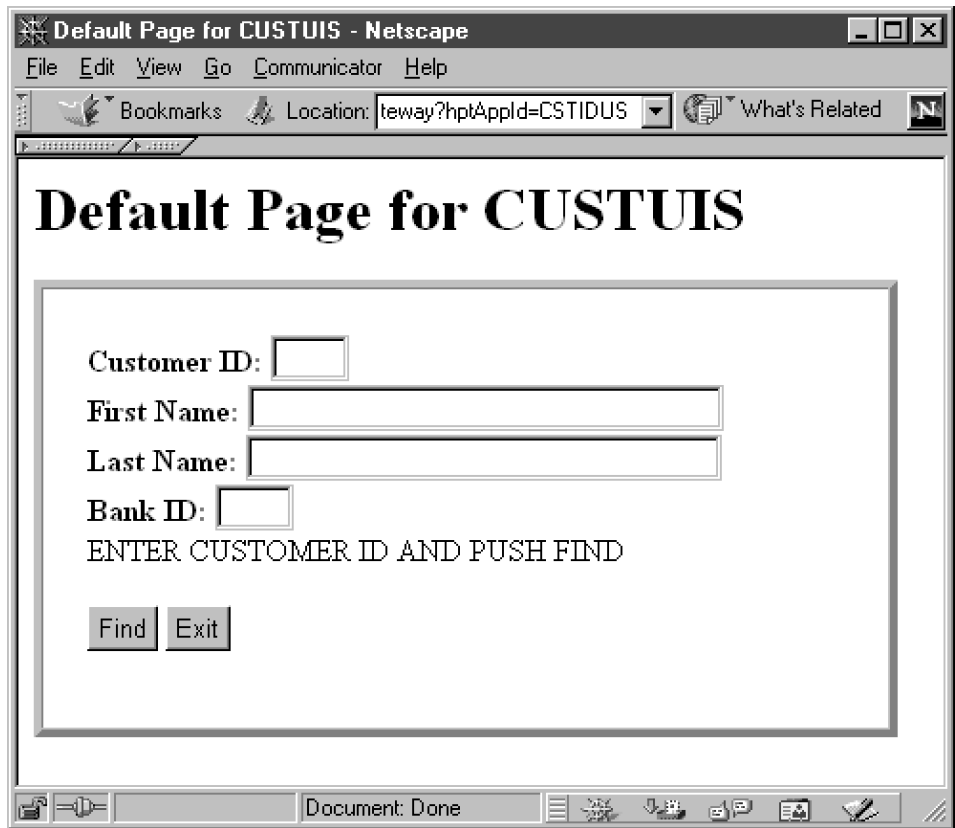


Figure 62. Customer Info Web page without the Update button

is not sent to the browser (the UI Type Hidden is sent, just not displayed). The data will not be seen in the browser or the raw HTML used to compose the browser view.

CSTIDUS-MAIN logic: The CSTIDUS-MAIN function logic includes the following:

- A loop around a CONVERSE statement to prompts for user input and to transfer control to the target web transaction program.
- The CSTIDUS-GET-PROFILE function is called using the CUSTUI_ID.USER data item as a parameter.
- The user type (update access) profile is stored in the STATEWS.UPDATE data item.
- Data is passed, as required, to the target web transaction program.

The CSTIDUS-MAIN function logic relevant for this review is as follows:

```
IF CUSTUI_ID.BUTTON-VALUE = 'OK' AND CUSTUI_ID.TARGET-APP = 'CNVUIREC';  
STATEWS.UPDATE = CSTIDUS-GET-PROF( CUSTUI_ID.USER); /* Get user profile  
DXFR CSTCNS STATEWS;          /* Transfer to target pgm - with profile data  
END;
```

Customer info web transaction: The CSTCNS web transaction program implements UI record-based state management using the CONVERSE model:

- Working storage STATEWS included in the **Specifications** list
- Record CSTPGMS-PARMS included in the **Tables and Additional Records** list
- Main function CSTCNS-MAIN performs a CONVERSE of UI record CUSTUIS.

The required function logic is shown in Figure 65 on page 97.

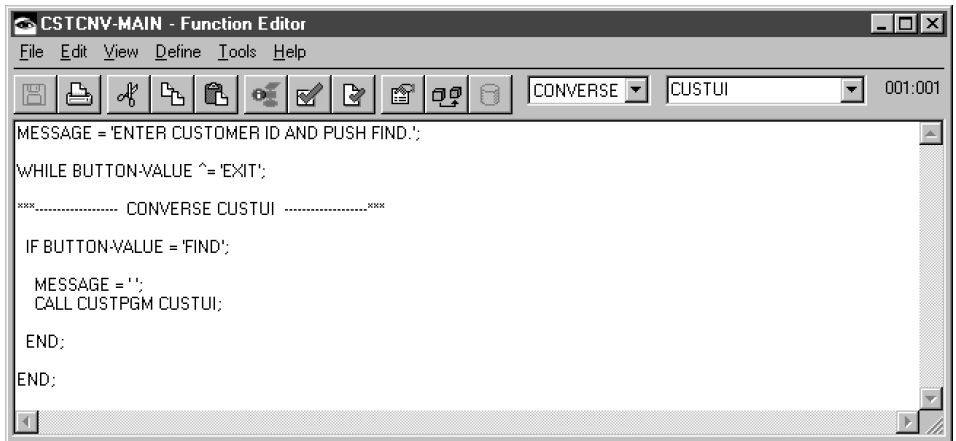


Figure 65. CSTCNS-MAIN processing logic (1)

The logic defines the following behavior:

- The server program is called as part of a loop around the CONVERSE.
- The loop continues until the Exit submit request is received.
- The server CUSTPGMS is called using the CUSTPGMS-PARMS working storage record as a parameter.
- The server program either returns data, updates data, or returns an error message in the MESSAGE field.

Use the test facility to test the CSTIDUS program. Choosing the **CNVUIREC** option will trigger invocation of the CSTCNS program. Check the UI record data item input values when interacting with the web transaction. Can you find the CANUPDATE data item in the browser or HTML? Try both **USER** and **MANAGER** values.

XFER with a named program

This model converts the CSTXP "XFER with a named program" web transaction into one which accepts state information from the CSTIDUS web transaction that identifies the user type and implements the state management by storing data in the UI record. The structure of this system is shown in Figure 66 on page 98.

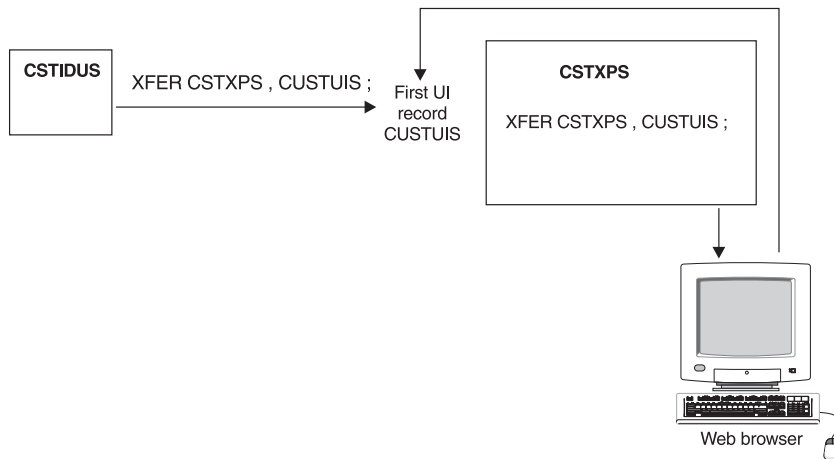


Figure 66. CSTXPS Web Transaction program structure

CSTIDUS-MAIN function logic: To support the CSTXPS program, this function passes the user type (update access) profile directly in the CANUPDATE data item in the CUSTUIS UI record. This data is then passed along to the CSTXPS web transaction program.

The CSTIDUS-MAIN function logic relevant for this exercise is as follows:

```

IF CUSTUI_ID.BUTTON-VALUE = 'OK' AND CUSTUI_ID.TARGET-APP = 'XPUIREC';
CUSTUIS.CANUPDATE = CSTIDUS-GET-PROF( CUSTUI_ID.USER); /* Get profile
CUSTUIS.BUTTONS[2] = ' ';
XFER CSTXPS ,CUSTUIS;          /* Transfer to target pgm - with profile data
END;
  
```

The state data is stored directly in the CUSTUIS UI record, so it is not accessible to the user. The data will not be in the HTML data sent to the browser, but stored as part of the session data kept for an XFER Pgm web transaction.

Customer info web transaction: The CSTXPS web transaction program implements the "XFER with a named program" model:

- UI Record CUSTUIS defined as the first UI record
- Record CSTPGMS-PARMS included in the list of **Tables and Additional Records**
- CSTXPS-MAIN is the associated main function.

The required function logic is shown in Figure 67.

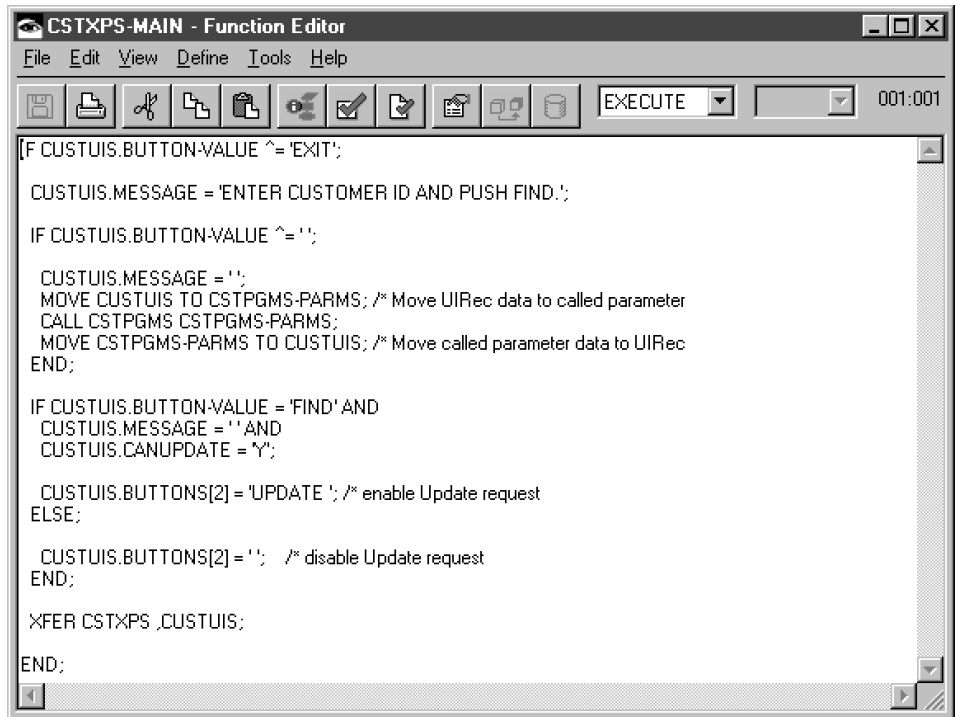


Figure 67. CSTXPS-MAIN processing logic

The logic defines the following behavior:

- Unless the **Exit** button is pushed, the predefined server program CUSTPGMS is called using the CUSTPGMS-PARMS UI record as a parameter.
- The server program either returns data, updates data, or returns an error message in the MESSAGE field.
- If a successful "Find" was performed and the user has update capability (provided through the CUSTUIS UI record) the **Update** button is displayed.
- To create a loop, the program invokes itself using an XFER Pgm statement.

Use the test facility to test the CSTIDUS program. Choosing the **XPUIREC** option will trigger invocation of the CSTXPS program. Check the UI record data item input values when interacting with the web transaction.

XFER with a blank

This section concerns an "XFER with a blank" web transaction that accepts state information from the CSTIDUS web transaction that identifies the user

type and implements the state management by storing data in the UI record. The structure of this system is shown in Figure 68.

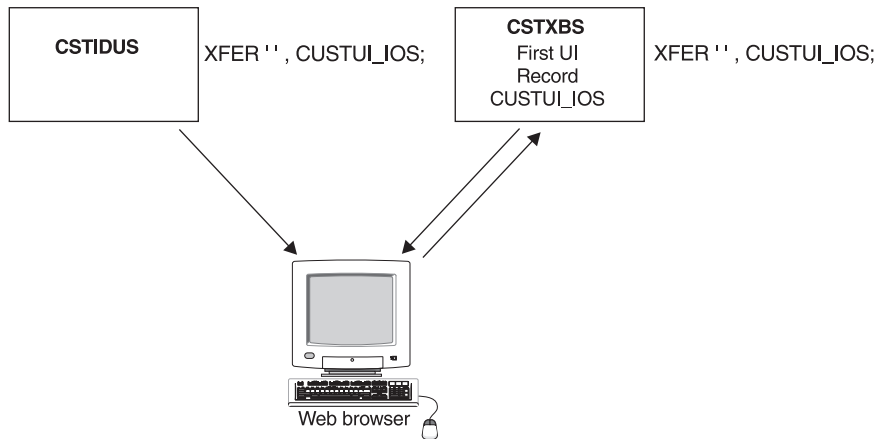


Figure 68. CSTXBS Web Transaction program structure

CSTIDUS-MAIN function logic: To support the CSTXBS program, this function passes the user type (update access) profile directly in the CANUPDATE data item in the CUSTUIS UI record. This data is then passed along to the CSTXPS web transaction program. The CSTIDUS-MAIN function logic relevant for this section is as follows:

```

IF CUSTUI_ID.BUTTON-VALUE = 'OK' AND CUSTUI_ID.TARGET-APP = 'XBUIREC';
CUSTUI_IOS.CANUPDATE = CSTIDUS-GET-PROF(CUSTUI_ID.USER); /* Get user profile
CUSTUI_IOS.BUTTONS[2] = ' ';
XFER '' ,CUSTUI_IOS; /* Transfer to target pgm - with profile data
END;
  
```

Input/output customer info UI record: The CUSTUI_IOS UI record is an edited copy of the CUSTUI_IO record:

- CUSTUI_IOS is defined as the first UI record.
- Defines a Program Link to program CSTXBS.

This definition will pass the state data forward to the target program. Without this program link parameter, the None UI Type value will not be passed to the target program.

The record definition is shown in Figure 69 on page 101.

The screenshot shows a window titled "CUSTUI_IOS - Record Editor" with a menu bar (File, Edit, View, Define, Tools, Help) and a toolbar. Below the toolbar is a dropdown menu for "User Interface" and a "Default Usage" dropdown set to "Nonshared". The main area contains a table with the following data:

Name	Occurs	Type	Length	Decimals	UI Type	UI Properties	Usage	Description
FORM IO	1	Char	171	0	Form	Custom	Nonshared	
BUTTON-VALUE	1	Char	8	0	None	None	Nonshared	
CUSTID	1	Char	4	0	Input/Output	Custom	Nonshared	
FNAME	1	Char	30	0	Input/Output	Custom	Nonshared	
LNAME	1	Char	30	0	Input/Output	Custom	Nonshared	
BANKID	1	Char	4	0	Input/Output	Custom	Nonshared	
BUTTON-SET	1	Char	24	0	Output	Default	Nonshared	
BUTTONS	2	Char	8	0	Submit	Custom	Nonshared	
BUTTON-EXIT	1	Char	8	0	Submit Bypass	Custom	Nonshared	
CANUPDATE	1	Char	1	0	None	None	Nonshared	
MESSAGE	1	Char	70	0	Output	Default	Nonshared	

Figure 69. CUSTUI_IOS UI record definition

Note: By moving the Buttons structure down in the UI record, we can move the buttons below the input fields in the defined form. Buttons are always at the bottom when a default form is created by VisualAge Generator, but with the defined form used in an XFER ' ' model, you control button placement.

Customer info web transaction: The CSTXBS web transaction program implements the "XFER with a blank" model:

- UI Record CUSTUI_IOS defined as the first UI record
- CSTPGMS-PARMS included in the **Tables and Additional Records** list
- CSTXBS-MAIN is the main function.

The required function logic is shown in Figure 70 on page 102.

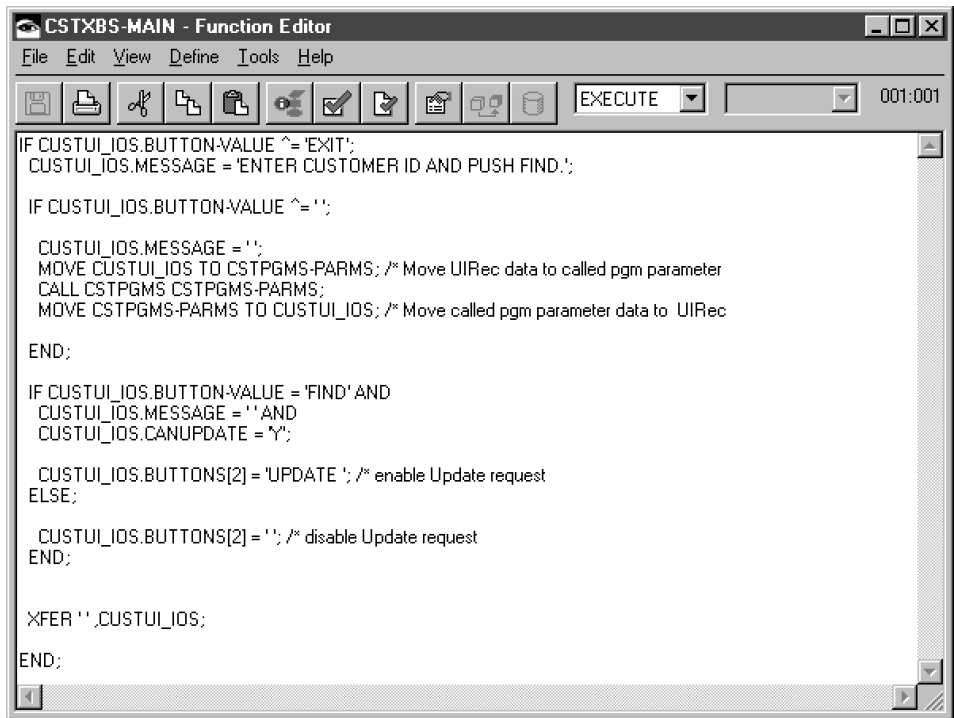


Figure 70. CSTXBS-MAIN processing logic

The logic includes the following functionality:

- Unless the **Exit** button is pushed, the predefined server program CUSTPGMS is called using the CUSTPGMS-PARMS UI record as a parameter. The server program either returns data, updates data, or returns an error message in the MESSAGE field.
- If a successful "Find" is performed and the user has update capability (provided through the CUSTUIS UI record), the **Update** button is displayed.
- To create a loop, the program invokes itself using an XFER ' ' statement.

Use the test facility to test the CSTIDUS program. Choosing the **XBUIREC** option will trigger invocation of the CSTXBS program. Set UI record watch points to monitor data.

Review the HTML sent to the browser. You should be able to find the CANUPDATE value, which, while defined as a None UI Type, becomes hidden data in the browser to implement the program link parameter required to pass this state data back to the target program.

If this data is sensitive, or you are concerned that the values might be manipulated, you might need to consider other options for state management when using the XFER ' ' program structure.

Working-storage-record-based state management

State data can be passed between web transactions using a working storage record, when the transfer command accepts a working storage record parameter (depends on the web transaction program structure). By using a working storage record, we avoid sending the data to both the WebSphere Application Server platform and the browser. This may be a requirement either when the data is sensitive or when the volume of data would delay network transport.

CONVERSE

The program CSTCNS2 was converted from the CSTCNS web transaction into one that uses a predefined state server program to obtain state data from an application database. The structure of this system is shown in Figure 71.

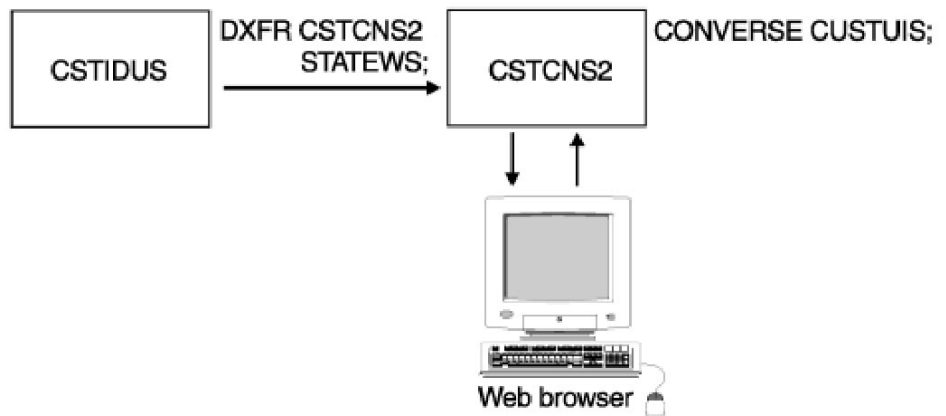


Figure 71. CSTCNS2 Web Transaction program structure

CSTIDUS-MAIN function logic: To support the CSTCNS2 program, this function passes the user type (update access) profile directly in the passed working storage record, where it will be kept during subsequent program logic.

The CSTIDUS-MAIN function logic relevant for this section is as follows:

```
IF CUSTUI_ID.BUTTON-VALUE = 'OK' AND CUSTUI_ID.TARGET-APP = 'CNVWKST';
STATEWS.UPDATE = CSTIDUS-GET-PROF( CUSTUI_ID.USER); /* Get user profile
DXFR CSTCNS2 STATEWS;
END;
```

Customer info web transaction: The CSTCNS2 web transaction program implements working-storage-record-based state management using the CONVERSE model:

- Working storage STATEWS included in the **Specifications** list
- CSTPGMS-PARMS included in the **Tables and Additional Records** list
- Main function CSTCNS2-MAIN performs a CONVERSE on UI record CUSTUIS.

The required function logic is shown in Figure 72.

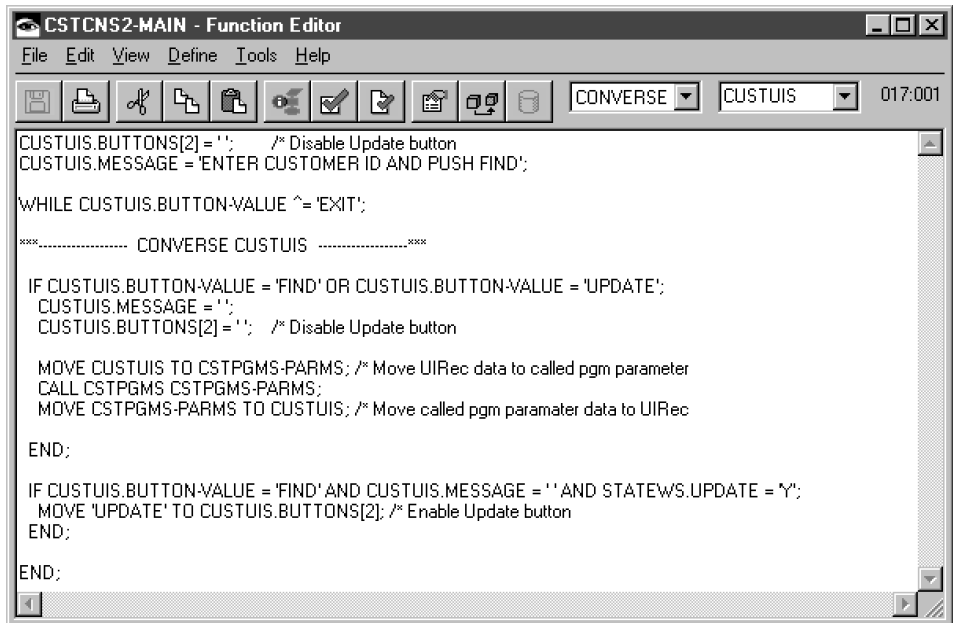


Figure 72. CSTCNS2-MAIN processing logic

The logic defines the following behavior:

- The server program is called as part of a loop around the Converse of the UI record.
- The loop continues until the Exit submit request is received.
- The predefined server program CUSTPGMS is called using the CUSTPGMS-PARMS UI record as a parameter.
- The server program either returns data, updates data, or returns an error message in the MESSAGE field.
- If a successful "Find" is performed and the user has update capability (provided through the STATEWS working storage record) the **Update** button is displayed.

Use the test facility to test the CSTIDUS program. Choosing the **CNVWKST** option will trigger invocation of the CSTCNS2 program. Use the test facility to check the UI record data item input values when interacting with the web transaction.

Note that the update profile information is no longer shown in the HTML code. Therefore, this is a very secure solution.

XFER with a named program

The web transaction program CSTXPS2 was created by converting the CSTXPS web transaction into one that accepts state information from the CSTIDUS web transaction and stores the state data in a working storage record.

CSTIDUS-MAIN function logic: To support the CSTXP2 program, this function passes the user type (update access) profile directly in the passed working storage record, where it will be kept during subsequent program logic.

The CSTIDUS-MAIN function logic relevant for this section is as follows:

```
IF CUSTUI_ID.BUTTON-VALUE = 'OK' AND CUSTUI_ID.TARGET-APP = 'XPWKST';
STATEWS.UPDATE = CSTIDUS-GET-PROF( CUSTUI_ID.USER); /* Get profile
CUSTUIS.BUTTONS[2] = ' ';
XFER CSTXPS2 STATEWS,CUSTUIS;
END;
```

Customer info web transaction: The CSTXPS2 web transaction program implements working-storage-record-based state management using the "XFER with a named program" model:

- UI Record CUSTUIS defined as the first UI record
- Working storage STATEWS included in the **Specifications** list
- CUSTPGMS-PARMS included in the **Tables and Additional Records** list
- CSTXPS2-MAIN is the main function.

The required logic is shown in Figure 73 on page 106.

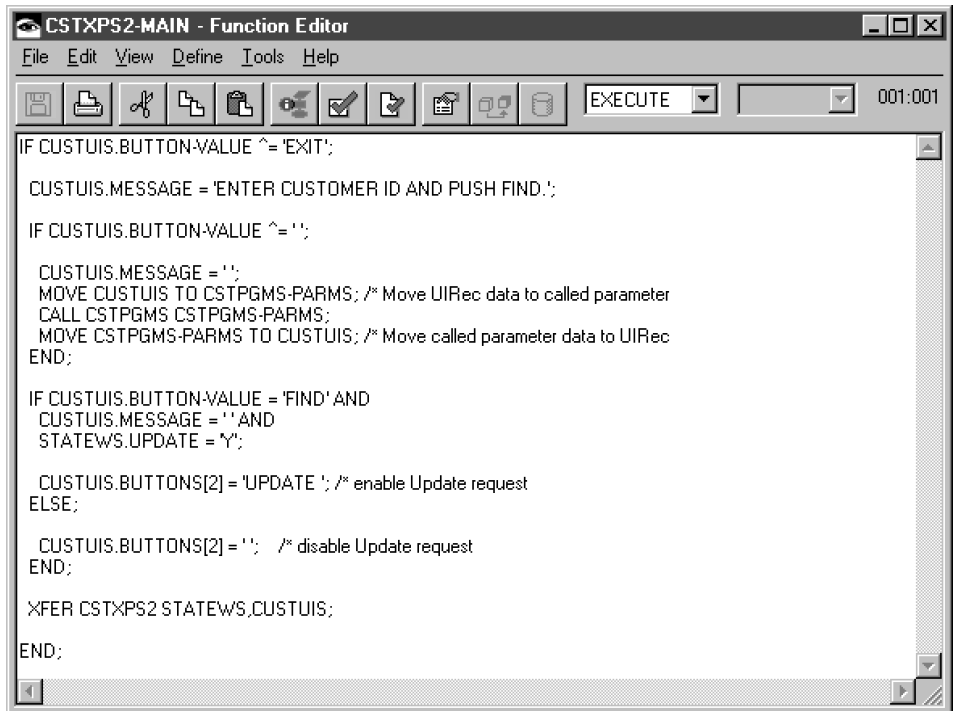


Figure 73. CSTXPS2-MAIN processing logic

The logic defines the following behavior:

- Unless the **Exit** button is pushed, the predefined server program CUSTPGMS is called using the CUSTPGMS-PARMS UI record as a parameter.
- The server program either returns data, updates data, or returns an error message in the **MESSAGE** field.
- If a successful "Find" is performed and the user has update capability (provided through the STATEWS working storage record), the **Update** button is displayed.
- To create a loop, the program invokes itself using an XFER Pgm statement.

Use the test facility to test the CSTIDUS program. Choosing the **XPWKST** option will trigger invocation of the CSTXPS2 program. Use the test facility to check the UI record data item input values when interacting with the web transaction.

Note: The update profile information is no longer shown in the HTML code. Therefore, this is a very secure solution.

Self-managed state implementation (XFER with a blank)

The XFER '' model does not support the use of a working storage record on the transfer request. This means that with the default behavior of VisualAge Generator XFER '' web transactions, the only way state data can be passed is in the UI record (where it will be visible in the HTML sent to the browser).

This may not satisfy some requirements either when the data is sensitive, or when the volume of data to be stored (when hidden in the HTML) would delay network transport.

Secure state data support in an XFER '' model web transaction must be implemented in the program logic.

This section involves conversion of the CSTXBS web transaction into one (CSTXBS2) that obtains state information from an application database, where it has been stored by the CSTIDUS web transaction. The structure of this system is shown in Figure 74.

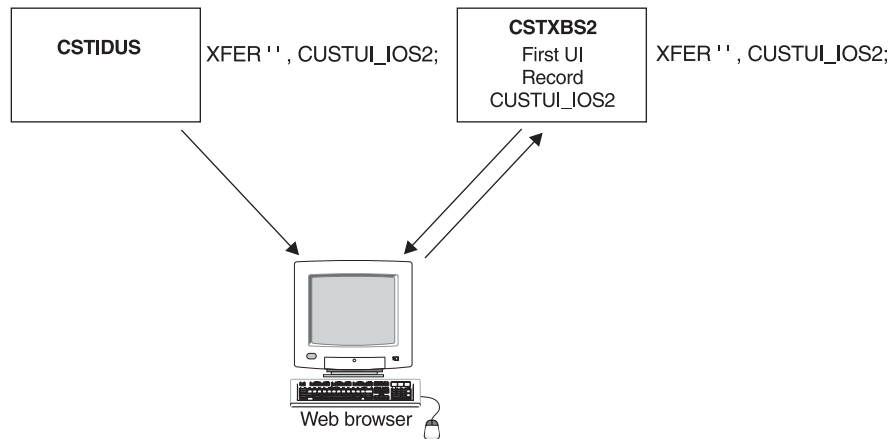


Figure 74. CSTXBS2 Web Transaction program structure

CSTIDUS-MAIN function and WTSPGM state management program

To support the CSTXB2 program, CSTIDUS-MAIN uses the predefined program WTSPGM to store the state information in an application database.

The WT_STATE_WS working storage record is passed to the WTSPGM program with the 'SAVE STATE' option stored in its WTS_ACTION data item. Access to the state data is obtained by passing the public part of the key in the data item CUSTUI_IOS2.BLIND-KEY. This will allow the target program (CSTXB2) to call WTSPGM to read the saved state data.

The CSTIDUS-MAIN function logic relevant for this section is as follows:

```
IF CUSTUI_ID.BUTTON-VALUE = 'OK' AND CUSTUI_ID.TARGET-APP = 'XBDB';

STATEWS.UPDATE = CSTIDUS-GET-PROF( CUSTUI_ID.USER); /* Get user profile
MOVE STATEWS TO WT_STATE_WS; /* Move state to server parm

WT_STATE_WS.WTS_ACTION = 'SAVE';
CALL WTSPGM WT_STATE_WS;      /* save state

IF WT_STATE_WS.WTS_KEY_TMSTMP ^= ' ';
MOVE WT_STATE_WS.WTS_KEY_TMSTMP TO CUSTUI_IOS2.BLIND-KEY;
CUSTUI_IOS2.BUTTONS[2] = ' ';
XFER ' ',CUSTUI_IOS2;
END;
MOVE "Problem saving state data" TO CUSTUI_ID.MESSAGE;

END;
```

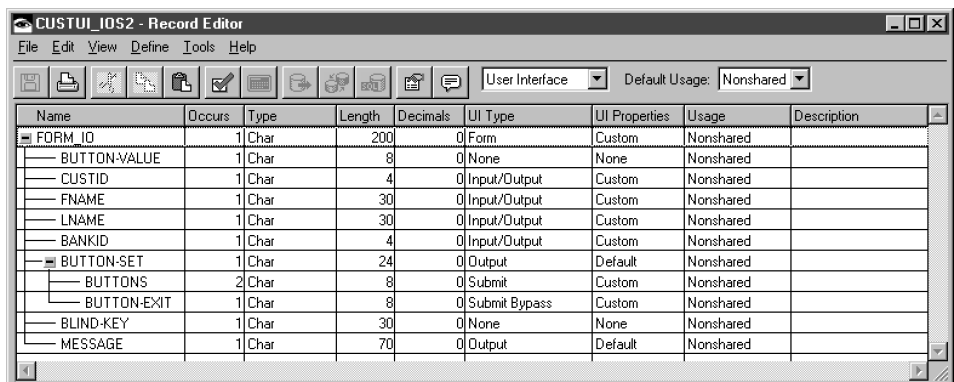
Input/output customer info UI record

The UI record CUSTUI_IOS2 is an edited copy of the CUSTUI_IOS UI record:

- Program CSTXBS2 defined as a Program Link
- CUSTUI_IOS2 defined as the first UI record
- An item with Name and Value Item of BLIND-KEY is included in the **Link Parameters** list

The passed value acts as a session identifier for the saved state data. This is safe to keep in the HTML, as it is not the actual state data.

The record definition is shown in Figure 75.



Name	Occurs	Type	Length	Decimals	UI Type	UI Properties	Usage	Description
FORM_ID	1	Char	200	0	Form	Custom	Nonshared	
BUTTON-VALUE	1	Char	8	0	None	None	Nonshared	
CUSTID	1	Char	4	0	Input/Output	Custom	Nonshared	
FNAME	1	Char	30	0	Input/Output	Custom	Nonshared	
LNAME	1	Char	30	0	Input/Output	Custom	Nonshared	
BANKID	1	Char	4	0	Input/Output	Custom	Nonshared	
BUTTON-SET	1	Char	24	0	Output	Default	Nonshared	
BUTTONS	2	Char	8	0	Submit	Custom	Nonshared	
BUTTON-EXIT	1	Char	8	0	Submit Bypass	Custom	Nonshared	
BLIND-KEY	1	Char	30	0	None	None	Nonshared	
MESSAGE	1	Char	70	0	Output	Default	Nonshared	

Figure 75. CUSTUI_IOS2 UI record definition

Customer info web transaction

The CSTXBS2 web transaction program implements self-managed state implementation using the "XFER with a blank" model:

- UI record CUSTUI_IOS2 defined as the first UI record
- Working storage WT_STATE_WS included in the **Specification** list
- Record CSTPGMS-PARMS included in the **Tables and Additional Records** list
- CSTXBS2-MAIN is the main function.

The logic is shown in Figure 76.

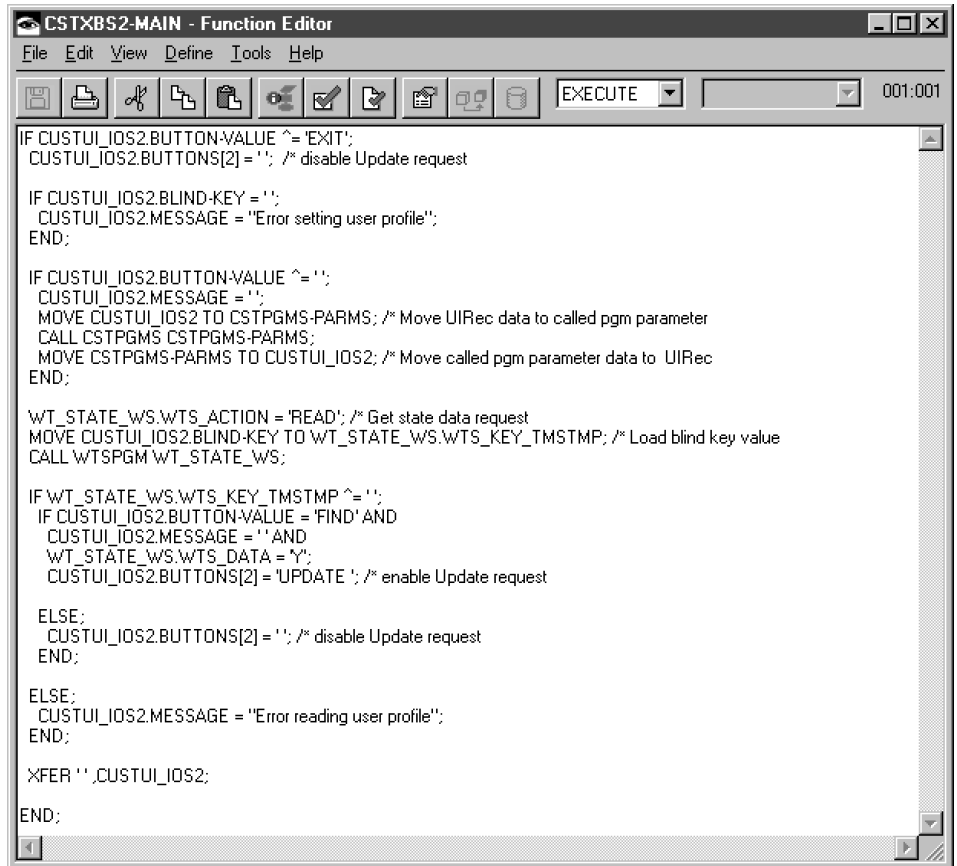


Figure 76. CSTXBS2-MAIN processing logic

The logic defines the following behavior:

- Unless the **Exit** button is pushed, the predefined server program CUSTPGMS is called using the CUSTPGMS-PARMS UI record as a parameter.
- The server program either returns data, updates data, or returns an error message in the MESSAGE field.

- If a successful "Find" is performed and the user has update capability, the **Update** button is displayed.
- The update capability is checked through the WTSPGM program, using the given WT_STATE_WS working storage record as a parameter. The 'READ STATE' option is stored in the WT_STATE_WS.WTS_ACTION data item; also, the BLIND-KEY data item is used to identify the session.
- To create a loop, the program invokes itself using an XFER ' ' statement.

Use the test facility to test the CSTIDUS program. Choosing the **XBDB** option will trigger invocation of the CSTXBS2 program.

Note: The exit logic should call the WTSPGM state management program with a DELETE request, but we will leave this up to you to add. In the meantime, the data left in the database will help you analyze your test runs. One option for implementing delete processing is to have the active web transaction delete all expired data, in order to have a self-cleaning system.

Chapter 7. Running Web Transactions in WebSphere Application Server

This chapter tells one way to set up a web application on Windows NT in WebSphere Application Server version 3.5, Advanced Edition (WAS) and how to run a generated web transaction there. The methods described in this chapter may not be appropriate for your environment, as different organizations set up web sites differently. If you are using WAS version 3.0, see the descriptions in the following document, which is cited in the Preface: *Building Enterprise Web Transactions using VisualAge Generator JavaBeans and JSPs*.

After installing WAS, add the Java Development Kit (JDK) to the *system* variable PATH, unless the JDK is already referenced there. For most installations, the added directory is as follows:

```
c:\WebSphere\AppServer\jdk\bin
```

Configuring a web app

By the procedures described later in this section, configure a web application:

1. Define an application server, servlet engine, and web application to WAS.
2. Configure the web application itself.
3. Complete the general set up—
 - a. Configure the gateway servlet.
 - b. Place the gateway-servlet JSPs where WAS can access them.
4. Install the generated tier 2 files for each web transaction.

Note: The instructions in the next sections assume that the web application is named `practice_app`.

Define an application server, servlet engine, and web application to WAS

To define an application server, servlet engine, and web application to WAS:

1. At the WebSphere Advanced Administrative Console dialog box, click **Console>Tasks>Create Application Server**.
2. The Types of Resources dialog box of the Create Application Server Wizard is displayed. Clear **Enterprise Beans**, select **Web Applications**, and click **Next**.
3. The Application Server Properties dialog box is displayed.
4. Name the application server (*VAgenAppServ*, for example).

5. At the command line arguments, specify a static classpath and include jar files `hpt` and `hptgateway`, as well as the directory where you want WAS to access your Java class files.

An example command line argument is as follows—

```
-classpath c:\ibmvagen\vgcso45\hpt.jar;  
c:\ibmvagen\vgwgs45\hptgateway.jar;c:\webclass1;
```

Make sure you include no spaces in the string that follows `-classpath`.

If you are using the CICS Transaction Gateway to interact with a remote CICS catcher, add the fully qualified path for files `ctgclient.jar` and `ctgserver.jar`, which are in the CICS installation `CLASSES` subdirectory.

6. In **Working Directory**, specify where WAS places output files (`c:\traces`, for example). This specification makes it easier to find WAS output.
7. Click **Next**.
8. The Application Server Start Options dialog box is displayed. **Select Do not start the server automatically after creating it** and click **Next**.
9. The Node Selection dialog box is displayed. Expand **Node**, select the machine name, and click **Next** until the Select Virtual Host dialog box is displayed.
10. Expand **Virtual Host Name**, select **default_host**, and click **Next**.
11. The Servlet Engine Properties dialog box is displayed. Name the Servlet Engine (*VAGenServletEngine*, for example).
12. The Web Application Properties dialog box is displayed.
13. Name the web application (*practice_app*, for example) and accept the value **default_host** for the virtual host.
14. In relation to the Web Application Web Path, specify the qualifier (*/Practice*, for example) that goes in the URL immediately after the host name. Click **Next**.
15. The Specify System Servlets dialog box is displayed. Select **Enable File Servlet**, clear **Serve Servlets by Classname**, and select **Enable JSP 1.0**.

Configure the web application

To configure the web application:

1. At the WebSphere Advanced Administrative Console dialog box, expand **WebSphere Administrative Domain**, the machine name, the application server (*VAGenAppServ*), and the servlet engine (*VAGenServletEngine*).
2. Click on the web application (*practice_app*).
3. In the rightmost screen (Web Application:practice_app), go to the Advanced tab. For Document Root, specify where you want WAS to access HTML files and JSPs (`c:\webroot1`). Specifying a root directory that

is near the top of the directory hierarchy makes it easier for you when, in VisualAge Generator, you need to specify where to place preparation output.

4. Right-click on the web application (*practice_app*) then **Create→Servlet**.
5. The Create Servlet dialog box, General tab is displayed.
6. For Servlet Name, type the following—
GatewayServlet
7. For Web Application, accept the value (*practice_app*).
8. For Servlet Class Name, type the following—
com.ibm.hpt.gateway.GatewayServlet
9. For Servlet Web Path List, type the URL used to access the gateway servlet—
default_host/Practice/GatewayServlet
10. Click the Advanced tab.
11. For Init Parameters, specify *hptGatewayProperties* as the only entry for Init Parm Name and specify a file name (*c:\linktabs\gw.properties*) as the related Init Parm Value. Specifying a value other than the default prevents you from overwriting the file when installing a new WAS modification.

For details on setting up gw.properties, see “Configuring a Gateway Servlet” on page 117.

Complete the general set up

To complete the general set up:

1. To configure the gateway in general, follow the directions in “Appendix A. Configuring the Gateway Servlet” on page 117.
2. Copy the gateway-servlet JSPs from the installation directory (usually, C:\Ibmvagen\Vgwgs45); the files are CSOERRORUIR.jsp, Vagen1EntryPage.jsp, Vagen1ErrorPage.jsp, and Vagen1LogonPage.jsp. Place those files in the document-root directory; in c:\webroot1, in this example.
3. Generate and prepare the tier 2 files for each web transaction included in the web application and install them, as described in “Deploy web-transaction files to tier 2” on page 60.

Running a web transaction in WAS

In WAS as in all tier 2 environments, do the following before running a generated web transaction:

1. Start the Session ID Manager.
2. Start the web application server.
3. Invoke the gateway servlet.

You also must start a catcher program—in most cases, a TCP/IP listener—on the third tier.

Start the Session ID Manager

Each ID created by the Session ID Manager is composed of a character prefix followed by a number and as many zeros as necessary to define an 8-character symbol. By default, the prefix is set to CUIR, and the available IDs range from CUIR0000 to CUIR9999. By starting the Session ID Manager with one argument, you can specify the prefix. The length of that prefix determines how many unique IDs are available.

To start the Session ID Manager on Windows NT or AIX, use the following command:

```
java com.ibm.hpt.gateway.SessionIDManager prefix
```

where *prefix* is an optional string of one to seven alphanumeric characters.

To start the Session ID Manager on OS/400, run the QVGNIMGR program.

To start the Session ID Manager on OS/390, take the following steps:

1. Create the following script:

```
export classpath=$classpath:/usr/lpp/vgwgs45/hpt.jar:/usr/lpp/vgwgs45/hptGateway.jar
java com.ibm.hpt.gateway.SessionIDManager
```

2. Run the following JCL code, where `sessionIDManager.sh` is the name of the script cited above:

```
//CSOIDMGR PROC
//SESSMGR EXEC PGM=BPXBATCH,REGION=4M
//STDIN DD PATH='/u/vguser/sessionIDManager.sh',
// PATHOPTS=(ORDONLY)
//STDOUT DD PATH='/u/vguser/sessionIDManagerStd.out',
// PATHOPTS=(OWRONLY,OCREAT),PATHMODE=SIRWXU
//STDERR DD PATH='/u/vguser/sessionIDManagerStd.err',
// PATHOPTS=(OWRONLY,OCREAT),PATHMODE=SIRWXU
```

Start the web application server

To start the web application server on WAS, do the following:

- From the WebSphere Advanced Administrative Console, select the web application server.
- Click the Run icon (a triangle on the tool bar).

Invoke the gateway servlet

To invoke the gateway servlet:

1. Open a browser.
2. Type the URL for the Gateway Servlet and press ENTER. The URL for the example—
`http://default_host/Practice/GatewayServlet`

3. The default entry page JSP presents the default VisualAge Generator System entry page.

Start a catcher program on tier 3

A catcher program must be running on each platform that has a web transaction. For details, see “hptLinkageProperties Parameter” on page 119.

Maintaining the work database on tier 3

If a work database is in use on tier 3, deleting old records from that database reduces the disk-storage requirements. Details are as follows:

- AIX** The work database is not in use for native AIX. The CICS for AIX environment uses a temporary storage queue.
- MVS** For details on maintaining the work database for OS/390 and IMS environments, see *VisualAge Generator Server Guide for MVS, VSE, and VM*. The CICS for MVS environment uses a temporary storage queue.

OS/400

During execution of a web transaction on OS/400, VisualAge Generator stores records in the work database VGNWORK. To remove old records from VGNWORK, execute the following command and include a specific date as a parameter—

```
CALL PGM(QVGNDBUT) PARM('yyyymmdd')
```

Records are deleted if they were written to VGNWORK before the specified date.

Solaris

The work database is not in use for native Solaris. The CICS for Solaris environment uses a temporary storage queue.

- VSE** For details on maintaining the work database, see *VisualAge Generator Server Guide for MVS, VSE, and VM*. The CICS for VSE environment uses a temporary storage queue.

Windows 2000, Windows NT

The work database is not in use for native Windows 2000 or Windows NT. The CICS for Windows NT environment uses a temporary storage queue.

Appendix A. Configuring the Gateway Servlet

To run VisualAge Generator web transactions, you must configure the VAGen gateway servlet on a web application server. The web application server must support Sun's Servlet 2.1 specification as well as the Java Server Pages specification 1.0.

Supported tier 2 environments are AIX, OS/390, OS/400, Solaris, Windows 2000, and Windows NT.

Configuring a Gateway Servlet

To configure the VisualAge Generator gateway servlet on the web application server, add a servlet that runs the `com.ibm.hpt.gateway.GatewayServlet` class. This class is in the `hptGateway.jar` file that is part of VisualAge Generator Server for Windows NT and should be added to the classpath. The following initial parameters should be set in order for the gateway servlet to operate properly.

Table 3. Gateway Servlet Parameters

Servlet Parameter	Parameter Value	Notes
<code>hptGatewayProperties</code>	<i>path name of gateway properties file</i>	Optional. You can optionally specify the gateway parameters in a file by using this parameter. Parameters specified in this file will override parameters specified in the servlet configuration.
<code>hptLogonPage</code>	<code>/Vagen1LogonPage.jsp</code>	Optional. If this parameter is omitted, no logon page is displayed. The entry page will be displayed. Requirements for the logon page are discussed below. <code>Vagen1LogonPage.jsp</code> is provided as a working example.
<code>hptEntryPage</code>	<code>/Vagen1EntryPage.jsp</code>	Either <code>hptEntryPage</code> or <code>hptEntryApp</code> is required. If both are specified, <code>hptEntryPage</code> will be used. This is a jsp that has a list of VAGen transactions that are available to run. <code>Vagen1EntryPage.jsp</code> is provided as a working example.

Table 3. Gateway Servlet Parameters (continued)

Servlet Parameter	Parameter Value	Notes
hptEntryApp	VAGen Web transaction name	Either hptEntryPage or hptEntryApp is required. If both are specified, hptEntryPage will be used. The name must correspond exactly to a definition in the VisualAge Generator Server environment, for example a CICS definition. There must also be an entry for it in the CSOGW.PROPERTIES file. This Web transaction will be invoked at startup, after logon (if logon is enabled), and when any Web transaction ends by EZECLDS.
hptErrorLog	path name of gateway servlet log file, for example, d:\server\log\VagenGateway.log; if this parameter is omitted, the default is not to create a log file	Optional. This is a log file for the VisualAge Generator gateway servlet. If specified, the gateway will log events and errors that can later be used by support to diagnose problems. Removing this parameter will turn off tracing.
hptLinkageProperties	path name of CSO gateway properties file	Required. The linkage table for client/server communications between the gateway servlet and the Web transaction. See "hptLinkageProperties Parameter" on page 119 for more information on these properties.
hptIDManagerHost	host name	Optional. This is the TCP/IP host name for the machine where the Session ID Manager process runs. The default is <i>localhost</i> .
hptDateMask	yyyy/dd/mm	Required if any User Interface Records have date fields. This parameter specifies the format for CHA date fields that are transmitted between the VAGen server application and the gateway. This should conform to the Long SYSGREGREN server parameter.
hptServletVersion	servlet version	Optional. The default is 2.1. Setting the parameter to 2.0 allows the gateway to work with Web servers that support only Servlet 2.0 specification and specification 0.91 JavaServer Pages.

hptLogonPage Parameter

The file Vagen1LogonPage.jsp is provided as an example of the logon page. There are several fields that are required in order for the logon page to work properly. The userid input field must have a name of hptUserId. The

password input field must have a name of hptPassword. Finally, the submit button must have a name of hptLogin. The userid and password are not validated until an application is selected to run.

hptEntryPage Parameter

The file Vagen1EntryPage.jsp is provided as an example of the entry page. There are several fields that are required in order for the entry page to work properly. The application name must be given in the field or select list with the name hptAppId. A field with the name of hptExec must also have a value. In the example, the Execute button has the name of hptExec.

An alternative to an entry page is an entry application. An entry application is a VisualAge Generator web transaction. When run, it can CONVERSE a UI record with links to other web transactions. It is also possible to have the application dynamically build a list of links to applications that are available based on identifying characteristics.

hptLinkageProperties Parameter

The hptLinkageProperties file tells the gateway servlet where to find the applications and how to communicate with them. This file provides the equivalent function of the linkage table in VisualAge Generator Client/Server programs. The hptLinkageProperties file contains server, application, and options entries.

Server Entry

The entries for the server section are in the form of:

`serverLinkage.servername.parameter=value`

servername

An arbitrary name which you assign to a set of values for a particular server. This name is used by the entries in the application entry.

parameter

One of the following:

commtype

Specifies the type of communications used to access the server program. The valid values are:

AS400	For use when your server program resides on an AS/400 system.
CICSECI	For use when your server program resides on an MVS, VSE, AIX, or CICS for Solaris system.
TCPIMS	For use when your server program resides on an IMS system.

TCPIP For use when your server program resides on a native Windows 2000, Windows NT, OS/2, AIX, HP-UX or Solaris system.

contable

Specifies the conversion table used on the server platform. The format is:

CS0zxxxx;

where z is the binary format, valid values are: I (Intel), E (EBCDIC) and X (UNIX), and xxxx is the code page of the target server machine. The SUN Java conversion routines are used to do the conversion. For more information on conversion tables for different server platforms and languages, see the *VisualAge Generator Client/Server Communications Guide*. The conversion is done according to the data definition in the UI Record.

location

For commtype=CICSECI, specifies the CICS system identifier which corresponds to the server name specified in the CICSCLI.INI file of the CICS Client. For commtype=TCPIMS, commtype=TCPIP, and commtype=AS400 specifies the TCP/IP host name of the server machine.

serverid

For commtype=CICSECI, specifies the CICS trans-id for the catcher. In most cases:

- The value of **serverid** is CPMI, which causes invocation of program DFHMIRS.
- If you specify a trans-id other than CPMI, CICS starts CPMI, which switches control to the transaction you specify.

To avoid starting CPMI when specifying a different CICS trans-id, prepend tpn_ to the trans-id.

For commtype=TCPIP, **serverid** specifies the port number of the listening socket on the server machine. The listening socket is configured by the CSOTCPUI executable on the native platforms and in the IMS TCPIP OTMA connection (ITOC) configuration on IMS. For commtype=AS/400, **serverid** is not used.

tcpport

For commtype=TCPIMS, specifies the port number of the listening socket on the server machine. The listening socket is configured in the IMS TCPIP OTMA connection (ITOC).

javaProperty

Specifies the java package where the UI record object and UI record bean for the web transaction are located. This entry is case sensitive.

imsgroupid

Specifies a security grouping in the ITOC configuration. This is only valid for TCPIMS connections.

imsdestid

Specifies the IMS system the request is sent to as defined in the ITOC configuration. This is only valid for TCPIMS connections.

library

Specifies the name of the library on the AS/400 that contains the server program. If you omit this entry or leave it blank, the server program is searched for in the QVGEN library followed by the library list specified by the system value of QUSRLIBL.

value

The value to which the parameter is being set.

Application Entry

The entries for the application section are in the form of:

`application.webtran=servername`

webtran

The name of the server program that you want to run. These are VisualAge Generator applications that you have generated to the target server environment.

The webtran value can use the * as a wildcard as the final character. For example, the entry `application.webui*=CICS5` will be used for all applications which start with the characters *webui*. If there are multiple application entries which match the server application, the most specific entry takes precedence. For example, if your `hptLinkageProperties` file contained the following entries:

```
application.webui*=abc
applicaiton.webuit*=def
```

and you wanted to run application *webuitran*, then the `serverLinkage.def` entries would be used.

servername

The value used to retrieve the `serverLinkage` properties. For example, if you were running server application `SERVER1`, and your application entry was:

```
application.SERVER1=CICS5
```

The serverLinkage properties would be retrieved from the entries specified as:

`serverLinkage.CICS5.parameter=value`

The application entries also allow you to override the serverLinkage entries for the application. For example, you can use this feature to specify different CICS transactions (serverid) for each application running on a CICS system without having to specify different serverLinkage entries for each one. Any parameter which is valid for the serverLinkage entry is also valid for the application entry. Any parameters being overridden must have the same webtran value as the value used for servername. The only exception to this rule is the traceFlag parameter described in “Options Entry”.

Options Entry

The following entries are valid for the options section:

`hptGateway.propertiesRefreshInterval=n`

- n* Specifies the number of minutes between refreshing of the hptLinkageProperties file in the gateway servlet.

This allows a dynamic update capability for adding additional server and application entries without having to stop and start the web application server. A value of 0 indicates that the properties file is never refreshed. Applications which have already been initiated will keep using the properties that were loaded at the time of initiation. Applications initiated after the refresh will use the new values.

`application.webname.traceFlag=n`

webname

The same as for the application entries including the wild card support.

- n* Can be either 0 or 1.

- 0** A value of 0 means that there is no communications layer tracing.
- 1** A value of 1 means that there is communications layer tracing for calls to the application(s) webtran. The communications layer tracing will only take place if there is an hptErrorLog entry specified in the gateway servlet configuration. The communications layer trace does not contain any usable information for the end user and should only be turned on if requested by IBM support.

Example hptLinkageProperties file

The following is a sample hptLinkageProperties file:

```
application.WEBUITRAN=CICS5
application.WEBUITRAN.serverid=WEBU
```

```
application.WEBUI*=CICS5
```

```

application.STF*=idaho

application.MATT*=IMSC
application.BASIC*=rtpas400

serverLinkage.CICS5.commttype=CICSECI
serverLinkage.CICS5.contable=CS0E037
serverLinkage.CICS5.location=nracics5
serverLinkage.CICS5.serverid=CPMI
serverLinkage.CICS5.javaProperty=my.pkg

serverLinkage.idaho.commttype=TCPIP
serverLinkage.idaho.contable=CS0X437
serverLinkage.idaho.location=machine01
serverLinkage.idaho.serverid=9877
serverLinkage.idaho.javaProperty=my.pak

serverLinkage.IMSC.commttype=TCPIMS
serverLinkage.IMSC.contable=CS0E037
serverLinkage.IMSC.location=carimsc
serverLinkage.IMSC.tcpport=4000
serverLinkage.IMSC.javaProperty=my.pak
serverLinkage.IMSC.msggroupid=
serverLinkage.IMSC.imsdestid=IMSC

serverLinkage.rtpas400.commttype=as400
serverLinkage.rtpas400.contable=CS0E037
serverLinkage.rtpas400.location=rtpas400
serverLinkage.rtpas400.javaProperty=my.pag
serverLinkage.rtpas400.library=sdearth

```

Configuring tier 3

Configuration information is included in this section for the following tier 3 platforms:

- CICS Systems
- Windows 2000, Windows NT, OS/2, AIX, HP-UX, and Solaris.
- IMS

Note: Supported tier 3 environments are those available for other generated programs, except that MVS TSO, OS/2 CICS, and VM are not supported.

CICS Systems: For CICS the catcher program is usually DFHMIRS, whose trans-id is CPMI, as shown in the hptLinkageProperties file displayed previously. The minimum TWA size for that trans-id is 1024 bytes.

Communications to CICS are by way of CICS Client ECI calls. When DFHMIRS is invoked, it is passed the name of the web transaction, which must be identified in the CICS tables.

For additional details, see the entry for **serverid** in “Server Entry” on page 119.

Windows 2000, Windows NT, OS/2, AIX, HP-UX, and Solaris: Configure CSO.INI file entry for WEBUI_SERVICE_NAME. The default value is VAGenWebUI. This specifies the service name that will be used to look up the TCP/IP port number that CSOTCPUI will listen on for incoming requests. This entry is case sensitive and must match an entry in the TCP/IP services file.

IMS: If the web transaction is running under IMS, set up the IMS TCP/IP OTMA Connection (ITOC), as described in *IMS e-business Connect Using the IMS Connectors* (SG24–5427–00), which is available from web site <http://www.redbooks.ibm.com>.

On IMS, the following parameters are required when defining ITOC:

```
APPLCTN PGMTYPE=TP,PSB=ims_psb_name
TRANSACT CODE=trancode,MODE=SNGL,EDIT=ULC
```

Note: Remote data is folded to uppercase if you omit the statement EDIT=ULC from the definition.

hptServletVersion Parameter

By default the VisualAge Generator gateway servlet supports Java web application servers that support JSP specification 1.0 and Servlet 2.1 support. gateway servlet also supports WebSphere 2.0 and other web application servers that support JSP 0.92 and Servlet 2.0. To run the gateway servlet with Servlet 2.0 support, set the hptServletVersion parameter to 2.0 in the VisualAge Generator gateway servlet properties file.

You do not need to make any changes for the generated UI record bean to be used with WebSphere 2.0 or above.

To use the generated JSPs with a web application server that supports only JSP specification 0.92, you must modify the generated JSPs as follows:

1. `<%@ page import = "com.ibm.vgj.uibean.VGDataElement" %>` becomes
`<%@ import="com.ibm.vgj.uibean.VGDataElement" %>`
2. `<jsp:useBean id="JANEUI1" scope="request" class="MY.PKG.JANEUI1Bean" />` becomes
`<BEAN NAME="JANEUI1" SCOPE=REQUEST TYPE="my.pkg.JANEUI1Bean"></BEAN>`

VisualAge Generator provides a conversion utility in the hptgateway.jar file. The command syntax to perform the conversion is as follows:

```
java ConvertJSP1toJSP92 filename
```

where *filename* is the .jsp file to be converted. The converted file will replace the existing file.

Appendix B. Transforming TUIs into Web Transactions

Existing Cross System Product (CSP) or VisualAge Generator text user interface (TUI) programs can be transformed into web transaction programs. This capability was designed to maximize the reuse of the existing business logic and therefore retain the investment made in current systems.

In this chapter we review these considerations and provide information on the processes that should be followed to perform this transformation. This is not a migration; there will be coding changes required to the logic in the web transaction program.

Considerations

To get started, you must first analyze what you intend to accomplish by making the current TUI system function available from the web, and decide whether you intend to add any additional functionality to the web transaction.

For example, suppose you need to quickly get the TUI transactions on the web, do not care about using the web UI functionality, and do you want to touch the code. In this case, you may want to consider using a screen scraping implementation such as CICS Host-On-Demand, CICS Web Interface with 3270 Bridge, or NetCICS. For more information see the following:

<http://www.ibm.com/software/ts/cics/library/whitepapers/cicsweb>

These alternative approaches would allow your CICS TUI transactions to become immediately available on the web. You may even consider this as a preliminary entry into the world of web-based applications.

If you want to take advantage of the usability features of the web such as radio buttons, drop down lists, forms, and program links, however, consider using the UI record feature to transform your TUI application into an e-business web transaction.

Phase 1 — analysis

This transformation effort should be approached as any other new web development. First, you should have an analysis phase to identify the functionality to be included in the web application.

The main purpose of this step is to determine what functions in the existing code are to be retained and to define what is required for screen navigation. This step can save converting unneeded program functions and serve to clarify end user interface components.

For example, a screen that displays a selection list of 20 codes or records and has forward, backward, right, and left scrolling, could be presented a web page that just does forward scrolling and handles the selection of a record. The code that handles the back, left, and right functions would be eliminated. A small (<100 items) single field selection list can be better presented as a drop-down list, eliminating even more code. Additional consideration will need to be given to how many items are shown in the list and how a selection in the list is made.

Phase 2 — basic transformation

Once you have identified the functions and code that will be used to implement a web transaction, you can begin the transformation.

The following list of tasks outlines the process of transforming a TUI program into a web transaction program with a web-based interface.

Prepare the code to be transformed

- Import your code into VisualAge Generator using the migration tool or import function, as described in the *VisualAge Generator Migration Guide*. There are additional considerations if you are moving from CSP 3.3 or earlier to VisualAge Generator for the first time, since going from an interpretive execution to generated COBOL may cause additional changes to be made to your source code. For details, see *Migrating Cross System Product Applications to VisualAge Generator*.
- Test using the TUI base code to ensure that all components are present and accounted for, as well to establish that the development environment is functioning properly. You can choose to implement the TUI base code at this time if this is your first VisualAge Generator implementation.
- Establish naming conventions and library organization (Projects, packages...) for your transformed parts and also for the new part types like the UI record and web transactions. Refer to *Guide to Migrating MSL's to ENVY*, SH23-0252 and the *VisualAge Generator Migration Guide*.

Convert the map to a UI record and replicate code

- For each user map in the VAGen parts browser, choose the map, then right-mouse click and select **Create UI Record from Map**. This will create a new VAGen UI record with fields for each named map field (variable field). Note that any label text or constant fields will not be included in the UI record.

The default label text will be the field name. Pay special attention to follow your naming standards for the new UI record name. It is recommended not to exceed the length of the name of the old field. For example, if the original map was named XY01M02 then a good name for the new UI record would be XY01U02. The reason for this suggestion is so you can more easily change the ESF syntax.

- Export the program using **VAGen Export with Associates**. Using your favorite text editor, globally change the map name to the UI record name where it is referenced in the code, change all occurrences of EZEMSG to EZMSG, and optionally, change other component names such as program name and function names (change the prefix to avoid conflicts with existing TUI parts). Good naming standards will simplify this step and reduce duplicate parts if you have both the old TUI program and the new web transaction in your workspace at the same time. Once the ESF changes have been made, import your new program from the changed ESF file.

Note: Once you are more comfortable with the web transaction programming model, you will be able to identify a level in your program structure where TUI logic may be reusable. For the purposes of an initial transformation, a complete replication of the existing code is recommended.

Implement UI record processing using CONVERSE

- Change the program type to web transaction.
- Assure that any CONVERSE functions are using the new UI record. If you have several maps (header, detail, trailer) that are presented by the DISPLAY option prior to the final CONVERSE, all the maps need to be merged into one UI record. Popup maps will have to be considered separately. Using a drop-down list or other UI functions may serve to replace their purpose.
- Add labels to each Input/Output field. UI record fields can be customized to add table, function, and other edits, as well as help text. Table edits will appear as drop-down lists. For function edits that are not doing I/O or server calls, then you have a choice where they are executed: on the web server or on the host server. Help text entered in for each field is included in the generated Java bean and can be referenced in the JSP, but is not referenced in the generated default JSP.
- Add submit buttons to your UI record for each function key processed and the ENTER key as well. You can set the initial value for each button to the EZE Aid equivalent value (ENTER, PF1, PF3...); however, a SET UI_record EMPTY clears the initial values. Submit buttons that have no value are not displayed. Therefore, set the submit values prior to the CONVERSE, as in the following example:

```

MOVE 'PF3' TO EXIT-BUTTON;
MOVE 'ENTER' TO SUBMIT-BUTTON;

***----- CONVERSE UGDETB-MAP1 -----***

IF EZEID NOT PF3;
...

```

Transform existing logic

- Statements that SET or TEST the TUI map item's attributes are not recognized and will cause errors to occur. For example, SET map item PROTECTED will be in error. These statements will need to be commented out or removed.
- How fields are displayed or hidden is left to the UI designer. In order to make this determination, additional data will need to be passed in the UI record.

For example, if your map has certain fields that are updatable depending on user security, then that user security information will need to be included in the UI record. The UI designer can then change the JSP to access the security data passed and set the display or hide fields appropriately.

- Testing a field for MODIFIED is allowed, but setting a field MODIFIED is not. This may also have an impact on your security or edit processing.
- Add a title to your UI record by editing the properties for the UI record itself.
- Change any date fields that are defined as numeric and have a date mask to be a character field (length 10). Then you must move EZEDTELC into this field if you want the date to be formatted.
- For map array fields that have been converted to items with occurrences, you need to add a counter item to the UI record to hold the number of elements to display in the list when the UI record is conversed. Then, in the custom settings for the item with occurrences, you need to specify the counter item as the occurrences item.

Test your program

- When a CONVERSE is reached, the browser is displayed with your UI record contents. You may need to further refine the code and UI record changes to complete the required baseline function.

Other things that may need special consideration in this new environment include:

- **Security** — Use of EZEUSR or application-specific security for read-only access versus update authority.
- **Attributes** — Setting of extended or highlight attributes — how this will be identified in the UI record. You could use a separate attribute field for each field, or a general one if possible.

- **Messages** — If you are using a Message Table, references to EZEMNO and EZEMSG will need to be changed to EZEUIERR. One advantage of this new environment is for multiple fields in error; all field level error messages are displayed below the field rather than one at a time in EZEMSG. General or informational messages like “Key selection and press enter” moved to EZEMSG will need to be moved to a UI record field like EZMSG instead. Be sure the text of the message still applies to the web interface.
- **Error checking** — If you currently have edit functions specified in your map, these will be included in the UI record bean you create from the map. Additionally, you can specify where the edits are to be performed on the web server or on the VAGen Server. However, if you SET map fields as MODIFIED to force the edit routine, then you will need to change the program to always execute the edit routine after the CONVERSE.
- **Navigation** — Consider uses of menus, “fastpath” fields, or other navigation. This function may be better implemented by the web designer.

Phase 3 — customization for the web

At this point your web transaction will still look like a TUI application that is displayed on a browser. You can enhance code to make the navigation and internal functionality more like other web-based applications. Among the possible additions:

- **Using Program Links as a record selection**— In the UI record, change a table column to be a type of program link, then customize it to specify the program and the first UI record that will be passed. Be sure that only key information is passed in this manner, because there is a limit to the amount of data that can be passed (around 400 bytes). The linked program must be prepared to handle the first UI record information and use the key information to read the detail record.
- *Use forms to combine multiple text screens into one UI record.* Because you are no longer limited to a 24x80 screen, you can possibly improve user productivity by retrieving more information on each request. Work with your web designer to identify what would be best for your users.
- *Consider using the XFER stateless model* for selection lists and inquiry-only programs, and stay with the CONVERSE model for maintenance tasks (insert, update, delete). To implement the XFER model, divide your single web transaction into two programs. Everything prior to the CONVERSE is in the first program, and everything after the CONVERSE is in the second program. Each program would use the same UI record. The UI record would have a parent field with UI Type of Form that links to the second program. Instead of the first program doing a CONVERSE, it would XFER ‘ ,UI_record. When the user selects a SUBMIT button, the second program is invoked and will evaluate the request and continue.

Some code is so intertwined with the map attributes and other settings, or so heavily modified, that it may be faster to use VisualAge Generator Templates to recreate the functionality of the program. This is especially true for simple (not much business logic) programs. For example, search for a record based on selection fields, then display a selection list that has a program link column to display or maintain the detail record.

Phase 4 — modify default JSP

Up to this point you have only seen the default Java Server Page (JSP) displayed in the browser. To improve the user presentation, this JSP will need to be enhanced with graphics and incorporated in existing web pages using web authoring tools such as WebSphere Studio. The UI designer can use the default JSP generated as a base or guide to create the customized JSP.

Index

Special Characters

<A> tag 26, 68
<FORM> structure
 and CONVERSE 11
 and UI type Form 19, 21, 22, 25
 default 21, 22, 23, 24, 35
 defined 4
 non-default 22, 24, 25

Numerics

3270 1, 42

A

AIX 4, 115, 117, 120, 123, 124
architecture 3
AS/400 119
Author-Time Values
 and error messages 54
 and generation option
 GENAUTHORTIMEVALUES 53
 and JSP invocation 54
 and realistic values 53
 defined 49, 53

B

bean
 defined 5
bean tag 65, 66, 124
benefit of web transaction xi, 1, 2, 3, 53
Boolean edit check, effect on
 HTML 28, 29
browser behavior
 and CONVERSE 12
 and XFER with a blank 16
 and XFER with a named
 program 14
browser session
 defined 4
browser window 12, 26

C

Callable ITF
 configuring the gateway
 servlet 52, 53
 defined 49, 50, 51
 setting up 51
catcher
 and Callable ITF 52
 and CICS 60, 112

catcher (*continued*)
 and WTE 62
 defined 9
checkbox 38, 40
CICS
 and catcher 60, 112, 123
 and hptLinkageProperties
 file 119, 120
 and not starting CPMI 120
 and password protection 6
 and screen scraping 127
 and TUI-program
 conversion 127
 and work database 115
CICS Transaction Gateway 60, 112
classpath
 in WAS 61, 112
 in WTE 55, 57, 60, 61
code page 120
communications layer tracing 122
complete state
 defined 2
 example 10, 11, 73, 74, 75, 76, 91,
 95, 96, 97, 103, 104, 105
control item
 occurrences 27
 selected index 27, 28, 31, 32, 34,
 35, 36, 37, 40
 submit value 27
controlled state
 defined 2
 example 12, 14, 76, 77, 78, 79, 81,
 82, 83, 97, 98, 99, 105, 107
conversation ID
 defined 7
 prefix 114
CONVERSE
 and browser behavior 12
 behavior of 10, 11, 12
 example of 10, 11, 12, 73, 74, 75,
 76, 91, 95, 96, 97, 103, 104, 105
conversion table 120
converting from TUI program 127,
 128, 129, 130, 131, 132
converting JSPs to level 0.92 124
country code 45
CPMI 9, 120, 123
Cross System Product 127, 128
CSO.INI 124

CSOERRORUIR 7, 47, 60, 113
CSOTCPUI 120, 124
CSP 127, 128
ctgclient 60, 112
ctgserver 60, 112

D

data bean 8
data editing
 input 16, 17, 43, 44
 output 8
date and time edits 43
DateFormat object 70
default
 resource bundle 45
deploying web-transaction files to
 tier 2 60, 61
directive in JSP 66
DISABLED 42
document root 60, 61, 112, 113
 setting in WTE 56
DXFR
 and callable ITF 51
 and commitment of
 resources 10

E

edit table
 defined 9
 naming convention 60
editing
 input 16, 17, 43, 44
 output 8
elementNamed() 68
entry page 6, 7, 60, 62, 113, 115, 119
environment variable
 HPTCLASSDIR 61
 HPTJSPDIR 61
error message 17
error page 7, 47
example web transactions 71, 73,
 74, 75, 76, 77, 78, 79, 81, 82, 83, 85,
 86, 88, 90, 91, 95, 96, 97, 98, 99, 100,
 101, 103, 104, 105, 107, 108, 111
extracting sample files 71
EZEAIID 27, 129
EZEDTELC 130
EZELTERM 7
EZEMNO 131
EZEMSG 129, 131

EZERGRGL 43
EZEUIERR 46, 47, 131
EZEUSR 7, 130
EZEUSRID 6
EZMSG 129

F

file names that include spaces 57
first UI record
 and data editing 17
 and submit value item 27
First UI record
 on Program Link tab 25
Form, UI type
 compared to Program Link 24, 25
 defined 19
formatting
 input 16, 17, 43, 44
 output 8

G

gateway servlet
 and browser session 4
 configuring
 for Callable ITF 52, 53
 defined 5
 prerequisites 117
 properties file
 and Callable ITF 53
 configuring tier 3 123
 defined 7, 119
 defining the application
 entry 121
 defining the options
 entry 122
 defining the server entry 119
 example 122
 setting up 117
 starting in WAS 114
 starting in WTE 62
GENAUTHORTIMEVALUES 53
generated files, naming
 convention 60
generation option
 GENAUTHORTIMEVALUES 53
 GENRESOURCEBUNDLE 46
 JAVADESTDIR 61
 JAVADESTHOST 61
 PACKAGENAME 61
 RESOURCEBUNDLELOCALE 46
 TARGNLS 46
GENRESOURCEBUNDLE 46
get<item name>() 68
getAppID() 68
getEditTableValues() 68

getErrorMessage() 68
getGatewayURL() 67, 68
getHelpText() 67, 69
getIndex() 69
getLabel() 69
getPageID() 68
getSecureGatewayURL() 68
getSessionID() 68
getTextValue() 69
getTextValuesTable() 69
getTitle() 67

H

hasInputError() 68, 69
help text 42
Hidden, UI type
 defined 19
HP-UX 120, 123, 124
HPTCLASSDIR 61
hptDateMask 43
hptEntryPage parameter 119
hptErrorLog 122
HPTJSPDIR 61
hptLinkageProperties 52
hptLinkageProperties file
 and Callable ITF 53
 configuring tier 3 123
 defined 7, 119
 defining the application
 entry 121
 defining the options entry 122
 defining the server entry 119
 example 122
hptLogonPage parameter 118
hptServletVersion 124
HTML
 <A> tag 26, 68
 <FORM> structure
 and CONVERSE 11
 and UI type Form 19, 21, 22, 25
 default 21, 22, 23, 24, 35
 defined 4
 non-default 22, 24, 25
 <PRE> structure 34, 35
 <SELECT> structure 34, 35, 36, 37
 <TABLE> structure 31, 32, 34, 37, 40
checkbox 28, 29
effect of a Boolean edit
 check 28, 29
effect of a match-valid table 34
effect of a selected index
 item 31, 32, 34, 35, 36, 37, 40

HTML (continued)

effect of a UI item array 30, 31, 34, 35
effect of data-item length 29, 30
effect of substructuring an
 array 37, 40
from item of UI type Input or
 Input/Output 28, 29, 30, 31, 32, 34, 42
from item of UI type Output 34, 35, 36, 37, 40, 42
from item of UI type Submit or
 Submit Bypass 41
 page structure 21, 23
HttpServletRequest 66
hypertext link 25, 26

I

IMS 119, 123
 and hptLinkageProperties
 file 120, 121
 and password protection 6
initAuthorTimeValues() 53, 54
Input, UI type
 defined 19
input editing 16, 17, 43, 44
Input/Output, UI type
 defined 20
interface bean 8
isDisplayable() 69
isEmpty() 69
isSelected() 69
ITF
 defined 49, 50
ITF Listener, for Callable ITF 51, 52
ITOC 120, 121, 124
ITSOBANK 71, 72

J

Java
 levels supported 63, 124
Java locale 9, 45, 46, 60
JAVADESTDIR 61
JAVADESTHOST 61
JSP
 bean tag 65, 124
 defined 5
 entry page 7, 60, 62, 113, 115
 error page 7, 47
 errorPage directive 67
 import directive 67, 124
 logon page 6, 7, 60, 113, 118, 119
 syntax 63, 64, 65, 66, 67
 UI record
 defined 8, 9
 invoking JSP directly 54

JSP (*continued*)

 naming convention 60

JSP specification 0.92, support
 for 124

L

language code 45

legacy program 1, 127, 128, 129,
 130, 131, 132

loading the sample code 72, 73

locale 9, 45, 46, 60

login page 6, 7, 60, 113, 118

M

match-valid table 34

message

 resource bundle 9, 45, 46, 60

 naming convention 60

 table 9, 46, 47

MVS 115, 123

MVS TSO 4

N

naming convention, generated
 files 60

None, UI type

 as a link parameter 40

 compared to Hidden 19

 defined 20, 40

O

occurrences() 69

occurrences item 27

Occurrences item 69

opening a new window 26

OS/2 120, 123, 124

OS/390 4, 114, 117

OS/400 4, 114, 115, 117, 120, 121

Output, UI type

 defined 20

output formatting 8

P

PACKAGENAME 61

page compilation 63

password 6

PrintWriter 63

Program Link, UI type

 compared to Form 24, 25

 defined 20

Program Link tab 25

program state

 alternatives 2, 91

 and CONVERSE 10, 11, 73, 74,
 75, 76, 91, 95, 96, 97, 103, 104,
 105

program state (*continued*)

 and XFER with a blank 14, 15,
 83, 85, 86, 88, 90, 99, 100, 101,
 103, 107, 108, 111

 and XFER with a named

 program 12, 14, 76, 77, 78, 79,
 81, 82, 83, 97, 98, 99, 105, 107

 complete state 2, 10, 11, 73, 74,
 75, 76, 91, 95, 96, 97, 103, 104,
 105

 controlled state 2, 12, 14, 76, 77,
 78, 79, 81, 82, 83, 97, 98, 99, 105,
 107

 defined 1

 stateless 2, 14, 15, 83, 85, 86, 88,
 90, 99, 100, 101, 103, 107, 108,
 111, 131

properties file

 and Callable ITF 53

 configuring tier 3 123

 defined 7, 119

 defining the application

 entry 121

 defining the options entry 122

 defining the server entry 119

 example 122

Q

QUSRLIBL 121

QVGEN 121

QVGND BUT 115

QVG NIMGR 114

R

radio button 37, 38

READONLY 42

request object 66

resource bundle

 default 45

 defined 5, 45

 effect of not having a specific
 one 45

 effect of not having any 46

 generation option

 GENRESOURCEBUNDLE 46

 RESOURCEBUNDLELOCALE 46

 TARGNLS 46

 message 9, 45, 46

 UI record 9, 45, 46

RESOURCEBUNDLELOCALE 46

S

screen scraping 127

scriptlet 63, 64, 65

security 6

segmented program

 defined 1

 described 1

selected index item 27, 28, 31, 32,
 34, 35, 36, 37, 40

servlet

 defined 5

Session ID Manager

 changing the ID prefix 114

 defined 7, 8

 starting in production 114

 starting in WTE 61

session object 66

set<item name>() 68

setDateTimeFormat() 43, 70

setErrMsg() 54

setting up the ITSOBANK

 database 71, 72

Solaris 4, 115, 117, 120, 123, 124

stateless

 defined 2

 example 14, 15, 83, 85, 86, 88, 90,
 99, 100, 101, 103, 107, 108, 111

 when converting TUI

 program 131

subElements() 69

Submit, UI type

 defined 20

 using an image 41, 42

 values and labels 40, 41

Submit Bypass, UI type

 defined 21

 using an image 41, 42

 values and labels 40, 41

submit value item 27

submitted form 25

T

TARGNLS 46

TCP/IP

 and Callable ITF 51

 and catcher 9, 120, 124

testing

 options for 49

text user interface 127, 128, 129,
 130, 131, 132

tier 1

 defined 3, 4

tier 2

 defined 3, 5, 6, 7, 8, 9

tier 3

 defined 3, 9, 10

toString() 64

tracing, communications layer 122

TUI program 127, 128, 129, 130,
 131, 132

U

- UI record
 - defined 2
 - resource bundle 9, 45, 46, 60
 - naming convention 60
 - testing modified items in 43
 - title 21
- UI record bean
 - defined 8
 - methods 67, 68, 69, 70
- UI record JSP
 - defined 8, 9
 - invoking directly 54
 - naming convention 60
- UI record object
 - defined 8
 - naming convention 60
- UI type
 - Form
 - compared to Program Link 24, 25
 - defined 19
 - Hidden
 - defined 19
 - Input
 - defined 19
 - Input/Output
 - defined 20
 - None
 - as a link parameter 40
 - compared to Hidden 19
 - defined 20, 40
 - Output
 - defined 20
 - Program Link
 - compared to Form 24, 25
 - defined 20
 - Submit
 - defined 20
 - using an image 41, 42
 - values and labels 40, 41
 - Submit Bypass
 - defined 21
 - using an image 41, 42
 - values and labels 40, 41
- useBean 65, 66, 124
- user connection ID
 - defined 7
 - prefix 114
- user think time 10
- userid 6

V

- VAGenWebUI 124
- variant 45

- VGDDataElement 66, 68, 69, 70
- VGNWORK 115
- VM 4, 123
- VSE 115

W

- WAS
 - and CICS 112
 - classpath 112
 - deploying web-transaction files 60, 61
 - setting up 111, 112, 113
 - starting the web application server 114
 - web application 111
- web application
 - defined 5, 54
 - in WAS 111
 - in WTE 54, 55, 56, 57
- web page
 - attribute setting 42, 43
 - cursor position 42
 - structure 21, 23
 - title 21
- web transaction examples 71, 73, 74, 75, 76, 77, 78, 79, 81, 82, 83, 85, 86, 88, 90, 91, 95, 96, 97, 98, 99, 100, 101, 103, 104, 105, 107, 108, 111
- WebSphere Application Server
 - as tier 2 3
- WEBUI_SERVICE_NAME 124
- Windows 2000 4
- Windows 2000, Windows NT 115, 117, 120, 123, 124
- Windows NT 4
- work database
 - defined 1
 - deleting old records 115
- WTE
 - and CICS 60
 - classpath 55, 57, 60
 - defined 49, 54
 - deploying web-transaction files 60, 61
 - document root 56
 - handling file names that include spaces 57
 - setting up 54, 55, 56, 57, 60, 61
 - starting 62
 - web application 54, 55, 56, 57

X

- XFER with a blank
 - and browser behavior 16
 - behavior of 10, 14, 15, 16, 40

- XFER with a blank (*continued*)

- example of 15, 83, 85, 86, 88, 90, 99, 100, 101, 103, 107, 108, 111
- XFER with a named program
 - and browser behavior 14
 - behavior of 10, 12, 14
 - example of 12, 76, 77, 78, 79, 81, 82, 83, 97, 98, 99, 105, 107



Printed in the United States of America
on recycled paper containing 10%
recovered post-consumer fiber.

SH23-0281-00

